

ANDROID APPLICATIONS DEVELOPMENT PRACTICAL APPROACH

Dr. John T Mesia Dhas

Dr. S. Naveen Kumar

Mr. D. Surendra

The Palm Series



ANDROID APPLICATIONS DEVELOPMENT PRACTICAL APPROACH

Dr. John T Mesia Dhas

Dr. S. Naveen Kumar

Mr. D. Surendra

The Palm Series



Title: ANDROID APPLICATIONS DEVELOPMENT PRACTICAL APPROACH

Author: Dr. John T Mesia Dhas, Dr. S. Naveen Kumar, D. Surendra

Publisher: Self-published by Dr. John T Mesia Dhas

Copyright © 2021 Dr. John T Mesia Dhas

All rights reserved, including the right of reproduction in whole or in part or any form

Address of Publisher: No-1, MGR Street, Charles Nagar, Pattabiram

Chennai – 600072

India

Email: jtdhasres@gmail.com

Printer: The Palm

Mogappair West

Chennai -600037

India

ISBN: 978-93-5445-403-5

Unit No	Topics	Page No
I INTRODUCTION TO ANDROID		
1	1.1 Introduction	1
	1.2 The Android 4.1 jelly Bean SDK	3
	1.3 Understanding the Android Software Stack	5
	1.4 Installing the Android	7
	1.5 Installing the Android SDK	9
	1.6 Creating Android Virtual Devices	13
	1.7 Creating the First Android Project	14
	1.87 Using the Text view Control	16
	1.9 Using the Android Emulator	18
	1.10 The Android Debug Bridge(ADB)	20
	1.11 Launching Android Applications on a Handset	21
	1.12 Creating an Android Project	24
II COMPONENTS OF ANDROID		
2	2.1 Understanding the Role of Android Application Components	27
	2.2 Understanding the Utility of Android API	30
	2.3 Overview of the Android Project Files	32
	2.4 Understanding Activities	34
	2.5 Role of the Android Manifest File	38
III BASIC CONTROLS		
3	3.1 Creating the User Interface	42
	3.2 Commonly Used Layouts and Controls	43
	3.3 Event Handling	44
	3.4 Displaying Messages Through Toast	44
	3.5 Creating and Starting an Activity	48
	3.6 Using the Edit Text Control	50
	3.7 Choosing Options with Checkbox	65
	3.8 Choosing Mutually Exclusive Items Using Radio Buttons	77
IV LAYOUTS AND RESOURCES		
4	4.1 Introduction to Layouts	91
	4.2 Linear Layout	92
	4.3 Relative Layout	106
	4.4 Absolute Layout	121

	4.5 Using Image View	125
	4.6 Frame Layout	127
	4.7 Table Layout	133
	4.8 Grid Layout	135
	4.9 Adapting to Screen orientation	142
	4.10 Creating Values Resources Using Drawable Resources	148
V EVENT DRIVEN CONTROLS		
5	5.1 Switching States with Toggle Buttons	156
	5.2 Creating an Images Switcher Application	165
	5.3 Scrolling Through Scroll View	172
	5.4 Playing Audio and Video	176
	5.5 All Displaying Progress with Progress Bar	187
VI MENUS		
6	6.1 Menus and Their Types	191
	6.2 Creating Menus Through XML	192
	6.3 Creating Menus Through Coding	211
	6.4 Applying a Context Menu to a List View	221
	6.5 Using the Action Bar	226
	6.6 Replacing a Menu with the Action Bar	237
	6.7 Creating a Drop-Down List Action Bar	246
VII TELEPHONY OPERATIONS		
7	7.1 Understanding Broadcast Receivers	251
	7.2 Using the Notification System	256
	7.3 Sending SMS Messages with Java Code	265
	7.4 Receiving SMS Messages,	274
	7.5 Sending Email	278
	7.6 Working With Telephony Manager.	285
VIII PRACTICAL PROGRAMS		
8	8.1 GUI Components	291
	8.2 Layout Managers and Event Listeners	295
	8.3 Calculator Application	301
	8.4 Graphical Primitives	306
	8.5 Database	309
	8.6 RSS Feed	316
	8.7 Multi-Threading	325
	8.8 GPS Location	330

	8.9 Writes data to the SD card	334
	8.10 Alerts	339
	8.11 Alarm Clock Application	342
IX IMPORTANT QUESTIONS		
9	9.1 Short Questions and Answers	347

CHAPTER 1

INTRODUCTION TO ANDROID

1.1 INTRODUCTION

When we talked about operating systems few years ago, the most common answers were Windows, Linux, and MAC operating system. However, with the undying competition in the mobile phones market, the next big thing entered was ANDROID, which in no time became the heart of smart phones. Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java language environment.

Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touch screen mobile devices such as smart phones and tablets. In addition, Google has further developed Android TV for televisions, Android Auto for cars, and Wear OS for wrist watches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.

The Android Operating System is a Linux-based OS developed by the Open Handset Alliance (OHA). The Android OS was originally created by Android, Inc., which was bought by Google in 2005. Google teamed up with other companies to form the Open Handset Alliance (OHA), which has become responsible for the continued development of the Android OS.

The android is a powerful operating system and it supports large number of applications in Smart phones. These applications are more comfortable and advanced for the users. The hardware that supports android software is based on ARM architecture platform. The android is an open source operating system means that it's free and any one can use it.

The android has got millions of apps available that can help you managing your life one or other way and it is available low cost in market at that reasons android is very popular.

Each time the OHA releases an Android version, it names the release after a dessert. Android 1.5 is known as Cupcake, 1.6 as Donut, 2.0/2.1 as Éclair, 2.2 as Froyo, 2.3 as Gingerbread, 3.0 to 3.2 as Honey Comb, 4.0 as Ice Cream Sandwich, 4.1 to 4.3 as Kit Kat, 5.0 to 5.1 as Lollipop, 6.0 as Marshmallow, 7.0 and 7.1 as Nougat, 8.0 and 8.1 as Oreo, 9 as pie, 10 as Android 10 and 11 as Android 11 (released September 2020).

THE IMPORTANT FEATURES OF ANDROID

- 1) The Android OS is designed for phones.
- 2) It is open-source.
- 3) Anyone can customize the Android Platform.
- 4) There are a lot of mobile applications that can be chosen by the consumer.
- 5) It provides many interesting features like weather details, opening screen, live RSS (Really Simple Syndication) feeds etc.
- 6) It provides support for messaging services(SMS and MMS), web browser, storage (SQLite), connectivity (GSM, CDMA, Blue Tooth, Wi-Fi etc.), media, handset layout etc.

Software developers who want to create applications for the Android OS can download the Android Software Development Kit (SDK) for a specific version. The SDK includes a debugger, libraries, an emulator, some documentation, sample code and tutorials. For faster development, interested parties can use graphical integrated development environments (IDEs) such as Eclipse to write applications in Java.

ANDROID EMULATOR

The Emulator is a new application in android operating system. The emulator is a new prototype that is used to develop and test android applications without using any physical device.



The android emulator has all of the hardware and software features like mobile device except phone calls. It provides a variety of navigation and control keys. It also provides a screen to display your application. The emulators utilize the android virtual device configurations. Once your application is running on it, it can use services of the android platform to help other applications, access the network, play audio, video, store and retrieve the data.

ANDROID VERSIONS

Google did not attach any high-calorie code name to its initial versions 1.0 and 1.1 of the Android Operating System. The code names of android ranges from A to N currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, Kit Kat, Lollipop, Marshmallow, Nougat, Oreo, Pie, android 10, android . Let's understand the android history in a sequence.



1.2 THE ANDROID 4.1 JELLY BEAN SDK

The Android 4.1 Jelly Bean SDK was released with new features for developers in July 2012. It improves the beauty and simplicity of Android 4.0 and is a major platform release that adds a variety of new features for users and app developers. A few of the big features of this release include the following:

- **Project Butter**—Makes the Jelly Bean UI faster and more responsive. Also CPU Touch Responsiveness is added, which increases CPU performance whenever the screen is touched. It uses the finger's speed and direction to predict where it will be located after some milliseconds, hence making the navigation faster.

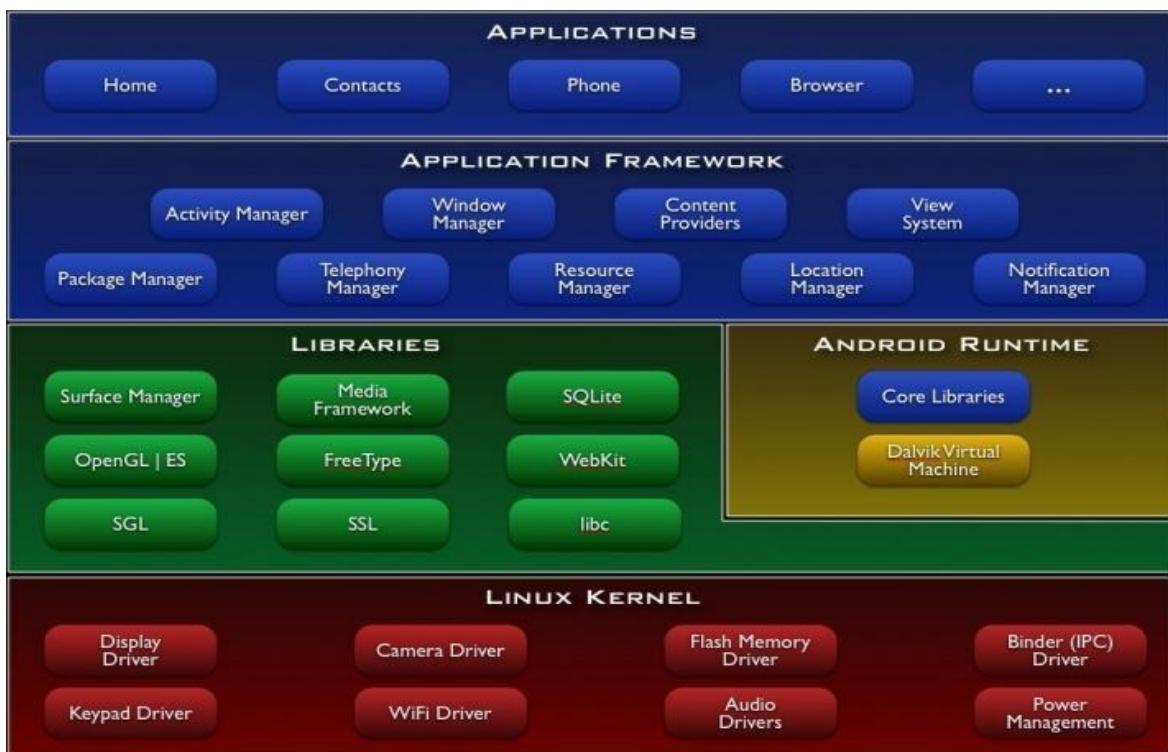
- **Faster speech recognition**—Speech recognition is now faster and doesn't require any network to convert voice into text. That is, users can dictate to the device without an Internet connection.
- **Improved notification system**— The notifications include pictures and lists along with text. Notifications can be expanded or collapsed through a variety of gestures, and users can block notifications if desired. The notifications also include action buttons that enable users to call directly from the notification menu rather than replying to email.
- **Supports new languages**—Jelly Bean includes support for several languages including Arabic, Hebrew, Hindi, and Thai. It also supports bidirectional text.
- **Predictive keyboard**—On the basis of the current context, the next word of the message is automatically predicted.
- **Auto-arranging Home screen**—Icons and widgets automatically resize and realign as per the existing space.
- **Helpful for visually impaired users**—The Gesture Mode combined with voice helps visually impaired users to easily navigate the user interface.
- **Improved Camera app**—The Jelly Bean Camera app includes a new review mode of the captured photos. Users can swipe in from the right of the screen to quickly view the captured photos. Also, users can pinch to switch to a new film strip view, where they can swipe to delete photos.
- **Better communication in Jelly Bean**—Two devices can communicate with Near Field Communication (NFC); that is, two NFC-enabled Android devices can be tapped to share data. Also, Android devices can be paired to Bluetooth devices that support the Simple Secure Pairing standard by just tapping them together.
- **Improved Google Voice search**—Jelly Bean is equipped with a question and answer search method that helps in solving users' queries similar to Apple's popular Siri.
- **Face Unlock**—Unlocks the device when the user looks at it. It also prevents the screen from blacking out. Optionally “blink” can be used to confirm that a live person is unlocking the device instead of a photo.
- **Google Now**—Provides users “just the right information at just the right time.” It displays cards to show desired information automatically. For example, the Places card displays nearby restaurants and shops while moving; the Transit card displays information on the next train or bus when the user is near a bus stop or railway station; the Sports card displays live scores or upcoming game events; the Weather card displays the weather conditions at a user's current location, and so on.
- **Google Play Widgets**—Provides quick and easy access to movies, games, magazines, and other media on the device. It also suggests new purchases on Google Play.
- **Faster Google Search**—Google Search can be opened quickly, from the lock screen and from the system bar by swiping up and also by tapping a hardware search key if it is available on the device.

- **Supports antipiracy**—This feature supports developers in the sense that the applications are encrypted with a device-specific key making it difficult to copy and upload them to the Internet.

1.3 UNDERSTANDING THE ANDROID SOFTWARE STACK / ANDROID ARCHITECTURE

The Android operating system is built on top of a modified Linux kernel. The software stack contains Java applications running on top of a virtual machine. Components of the system are written in Java, C, C++, and XML. Android operating system is a stack of software components which is roughly divided into five sections

- 1) Linux kernel
- 2) Native libraries (middleware),
- 3) Android Runtime
- 4) Application Framework
- 5) Applications



1) Linux kernel

It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access. This layer is the foundation of the Android Platform.

- Contains all low level drivers for various hardware components support.
- Android Runtime relies on Linux Kernel for core system services like,

- Memory, process management, threading etc.
- Network stack
- Driver model
- Security and more.

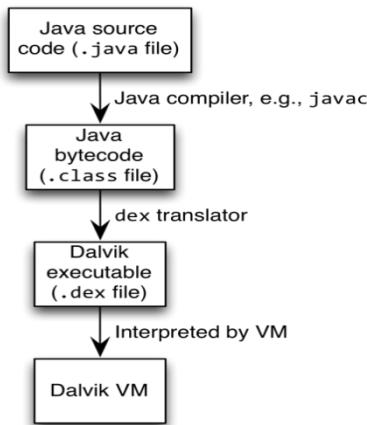
2) Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

- **SQLite** Library used for data storage and light in terms of mobile memory footprints and task execution.
- **WebKit** Library mainly provides Web Browsing engine and a lot more related features.
- The **surface manager** library is responsible for rendering windows and drawing surfaces of various apps on the screen.
- The **media framework** library provides media codecs for audio and video.
- The **OpenGL** (Open Graphics Library) and **SGL**(Scalable Graphics Library) are the graphics libraries for 3D and 2D rendering, respectively.
- The **FreeType** Library is used for rendering fonts.

3) Android Runtime

In android runtime, there are core libraries and DVM (Dalvik Virtual MACHine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual MACHine.



- Dalvik is a specialized virtual Machines designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.
- Android apps execute on Dalvik VM, a “clean-room” implementation of JVM
- Dalvik optimized for efficient execution
- Dalvik: register-based VM, unlike Oracle’s stack-based JVM

- Java .class bytecode translated to Dalvik EXecutable (DEX) bytecode, which Dalvik interprets

4) Android Framework

On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

- **Activity Manager:** manages the life cycle of an applications and maintains the back stack as well so that the applications running on different processes has smooth navigations.
- **Package Manager:** keeps track of which applications are installed in your device.
- **Window Manager :** Manages windows which are java programming abstractions on top of lower level surfaces provided by surface manager.
- **Telephony Managers:** manages the API which is use to build the phone applications
- **Content Providers:** Provide feature where one application can share the data with another application. like phone number , address, etc
- **View Manager :** Buttons , Edit text , all the building blocks of UI, event dispatching etc.

5) Applications

- On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernel. Any applications that you write are located at this layer.

1.4 INSTALLING THE ANDROID

1) First of all, Download android studio from this link:
<https://developer.android.com/studio/index.html>

2) JDK 8 is required when developing for Android 5.0 and higher (JRE is not enough). To check if you have JDK installed (and which version), open a terminal and type javac -version. If the JDK is not available or the version is lower than 6, download it from this link.

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

3) To set up Android Studio on **Windows**:

1.Launch the .exe file you just downloaded.

2.Follow the setup wizard to install Android Studio and any necessary SDK tools.

On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.

Select Start menu > Computer > System Properties > Advanced System Properties. Then open Advanced tab > Environment Variables and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.8.x.(where x is version number).

To set up Android Studio on MAC OS:

1. Launch the .dmg file you just downloaded.
2. Drag and drop Android Studio into the Applications folder.
3. Open Android Studio and follow the setup wizard to install any necessary SDK tools.

Depending on your security settings, when you attempt to open Android Studio, you might see a warning that says the package is damaged and should be moved to the trash. If this happens, go to System Preferences > Security & Privacy and under Allow applications downloaded from, select Anywhere. Then open Android Studio again.

If you need use the Android SDK tools from a command line, you can access them at:

/Users/<user>/Library/Android/sdk/ To set up Android Studio on **Linux**:

- 1.Unpack the downloaded ZIP file into an appropriate location for your applications.
- 2.To launch Android Studio, navigate to the android-studio/bin/ directory in a terminal and execute studio.sh. You may want to add android-studio/bin/ to your PATH environmental variable so that you can start Android Studio from any directory.
- 3.Follow the setup wizard to install any necessary SDK tools.

Android Studio is now ready and loaded with the Android developer tools, but there are still a couple packages you should add to make your Android SDK complete.

- 4) The SDK separates tools, platforms, and other components into packages you can download as needed using the Android SDK Manager. Make sure that you have downloaded all these packages.

To start adding packages, launch the Android SDK Manager in one of the following

ways:

- In Android Studio, click SDK Manager in the toolbar.
- If you're not using Android Studio:

Windows: Double-click the SDK Manager.exe file at the root of the Android SDK directory.

MAC/Linux: Open a terminal and navigate to the tools/ directory in the Android SDK, then execute android sdk.

5) Now get all the SDK tools, support libraries for additional APIs, Google Play services (if you need to use them). 6)Once you've selected all the desired packages, continue to install:

Click Install X packages.

In the next window, double-click each package name on the left to accept the license agreement for each. Click Install.

The download progress is shown at the bottom of the SDK Manager window. Do not exit the SDK Manager or it will cancel the download.

7)Now that we have downloaded and installed everything we need. Enjoy your experience with Android. Best of luck from Internshala.com

1.5 INSTALLING THE ANDROID SDK

For developing native Android applications that you can publish on the Google Play marketplace, you need to install the following four applications:

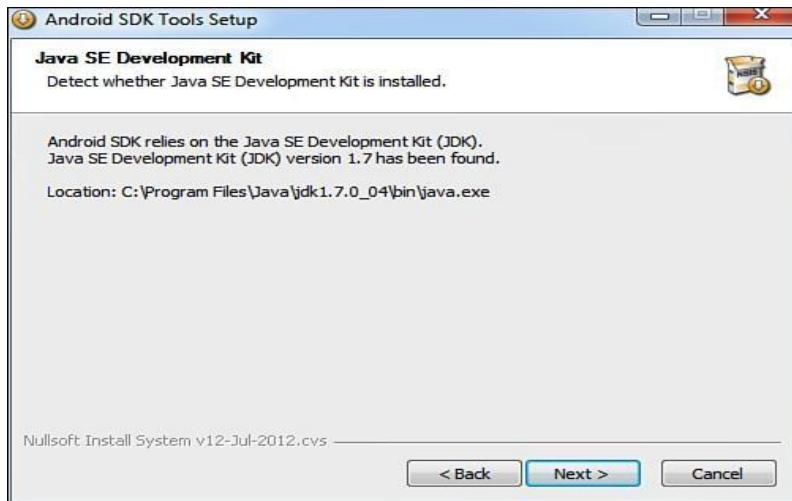
- **The Java Development Kit (JDK)** can be downloaded from <http://oracle.com/technetwork/java/javase/downloads/index.html>.
- **The Eclipse IDE** can be downloaded from <http://www.eclipse.org/downloads/>.
- **The Android Platform SDK Starter Package** can be downloaded from <http://developer.android.com/sdk/index.html>.
- **The Android Development Tools (ADT) Plug-in** can be downloaded from <http://developer.android.com/sdk/eclipse-adt.html>. The plug-in contains project templates and Eclipse tools that help in creating and managing Android projects.

The Android SDK is not a full development environment and includes only the core SDK Tools, which are used to download the rest of the SDK components. This means that after

installing the Android SDK Tools, you need to install Android platform tools and the other components required for developing Android applications. Go to <http://developer.android.com/sdk/index.html> and download the package by selecting the link for your operating system.

The first screen is a Welcome screen. Select the Next button to move to the next screen. Because the Android SDK requires the Java SE Development Kit for its operation, it checks for the presence of JDK on your computer.

If Java is already installed on your computer before beginning with Android SDK installation, the wizard detects its presence and displays the version number of the JDK



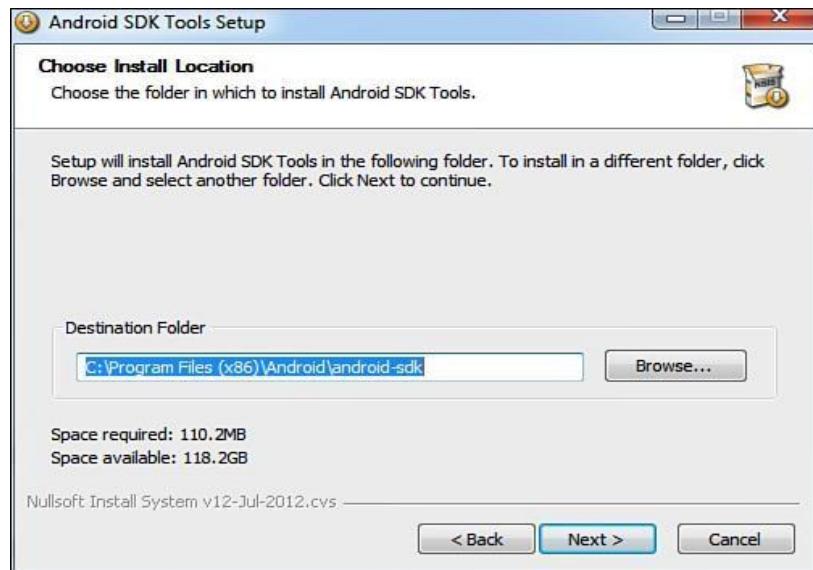
found on the Machine, as shown in Figure.

Figure Dialog box informing you that the JDK is already installed on the computer

Select the Next button. You get a dialog box asking you to choose the users for which Android SDK is being installed. The following two options are displayed in the dialog box:

- Install for anyone using this computer
- Install just for me

Select the Install for anyone using this computer option and click Next. The next dialog prompts you for the location to install the Android SDK Tools, as shown in below Figure . The dialog also displays the default directory location for installing Android SDK Tools as C:\Program Files (x86)\Android\android-sdk, which you can change by selecting the Browse button. Keep the default directory for installing Android SDK Tools unchanged; then select the Next button to continue.



The next dialog box asks you to specify the Start Menu folder where you want the program's shortcuts to appear, as shown.



Figure: Dialog box to select the Start menu shortcut folder

A default folder name appears called Android SDK Tools. If you do not want to make a Start Menu folder, select the Do not create shortcuts check box. Let's create the Start Menu folder by keeping the default folder name and selecting the Install button to begin the installation of the Android SDK Tools. After all the files have been downloaded and installed on the computer, select the Next button. The next dialog box tells you that the Android SDK Tools Setup Wizard is complete and the Android SDK Tools have successfully installed on the computer. Select Finish exiting the wizard, as shown in Figure



Figure: Successful installation of the Android SDK Tools dialog box

Note that the check box Start SDK Manager (to download system images) is checked by default. It means that after the Finish button is clicked, the Android SDK Manager, one of the tools in the Android SDK Tools package, will be launched. Android SDK is installed in two phases. The first phase is the installation of the SDK, which installs the Android SDK Tools, and the second phase is installation of the Android platforms and other components. An Android application is a combination of several small components that include Java files, XML resource and layout files, manifest files, and much more. It would be very time-consuming to create all these components manually. So, you can use the following applications to help you:

- **Eclipse IDE**—An IDE that makes the task of creating Java applications easy. It provides a complete platform for developing Java applications with compiling, debugging, and testing support.
- **Android Development Tools (ADT) plug-in**—A plug-in that's added to the Eclipse IDE and automatically creates the necessary Android files so you can concentrate on the process of application development.

Before you begin the installation of Eclipse IDE, first set the path of the JDK that you installed, as it will be required for compiling the applications. To set the JDK path on Windows, right-click on My Computer and select the Properties option. From the System Properties dialog box that appears, select the Advanced tab, followed by the Environment Variables button. A dialog box, Environment Variables, pops up. In the System variables section, double-click on the Path variable. Add the full path of the JDK (C:\Program Files\Java\jdk1.7.0_04\bin\java.exe) to the path variable and select OK to close the windows.

1.5 CREATING ANDROID VIRTUAL DEVICES

An Android Virtual Device (AVD) represents a device configuration. There are many Android devices, each with different configuration. To test whether the Android application is compatible with a set of Android devices, you can create AVDs that represent their configuration. For example, you can create an AVD that represents an Android device running version 4.1 of the SDK with a 64MB SD card. After creating AVDs, you point the emulator to each one when developing and testing the application. AVDs are the easiest way of testing the application with various configurations.

To create AVDs in Eclipse, select the Window, AVD Manager option. An Android Virtual Device Manager dialog opens, as shown in Figure. The dialog box displays a list of existing AVDs, letting you create new AVDs and manage existing AVDs. Because you haven't yet defined an AVD, an empty list is displayed.

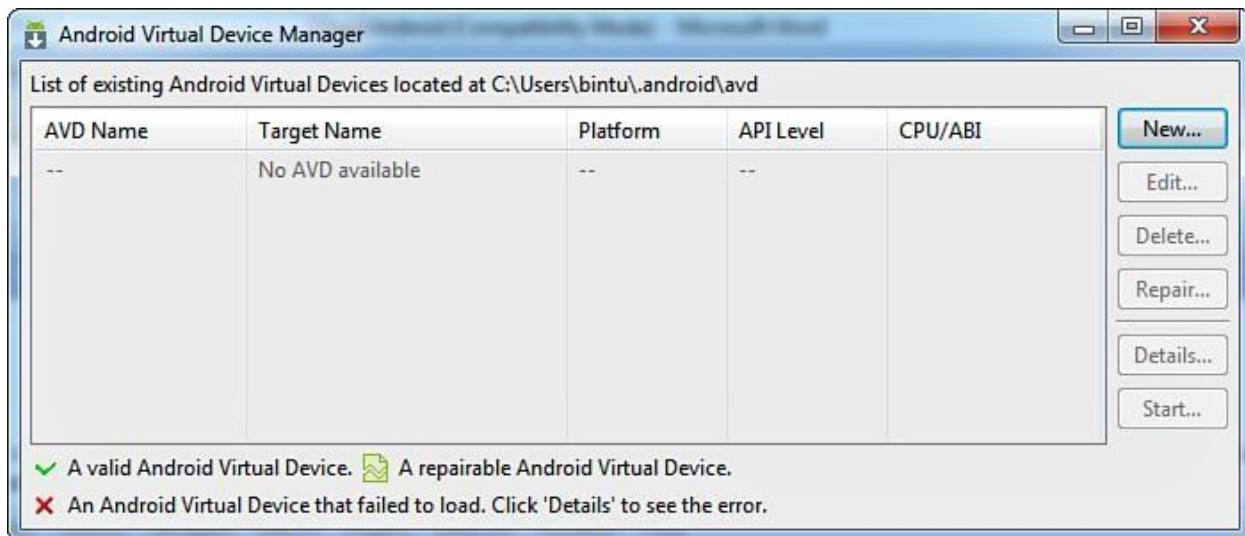


Figure: The AVD Manager dialog

Select the New button to define a new AVD. A Create new Android Virtual Device (AVD) dialog box appears. The fields are as follows:

- Name—Used to specify the name of the AVD.
- Target—Used to specify the target API level. Our application will be tested against the specified API level.
- CPU/ABI—Determines the processor that we want to emulate on our device.
- SD Card—Used for extending the storage capacity of the device. Large data files such as audio and video for which the built-in flash memory is insufficient are stored on the SD card.
- Snapshot—Enable this option to avoid booting of the emulator and start it from the last saved snapshot. Hence, this option is used to start the Android emulator quickly.
- Skin—Used for setting the screen size. Each built-in skin represents a specific screen size.

You can try multiple skins to see if your application works across different devices.

- **Hardware**—Used to set properties representing various optional hardware that may be present in the target device.

In the AVD, set the Name of the AVD to demoAVD, choose Android 4.1—API Level 16 for the Target, set SD Card to 64 MiB, and leave the Default (WVGA800) for Skin.

In the Hardware section, three properties are already set for you depending on the selected target. The Abstracted LCD density is set to 240; the Max VM application heap size is set to 48, and the Device RAM size is set to 512.

You can select these properties and edit their values, delete them, and add new properties by selecting the New button. New properties can include Abstracted LCD density, DPad support, Accelerometer, Maximum horizontal camera pixels, Cache partition size, Audio playback support, and Track-ball support, among others.

Note

The larger the allocated SD Card space, the longer it takes to create the AVD. Unless it is really required, keep the SD Card space as low as possible. I would recommend keeping this small, like 64MiB.

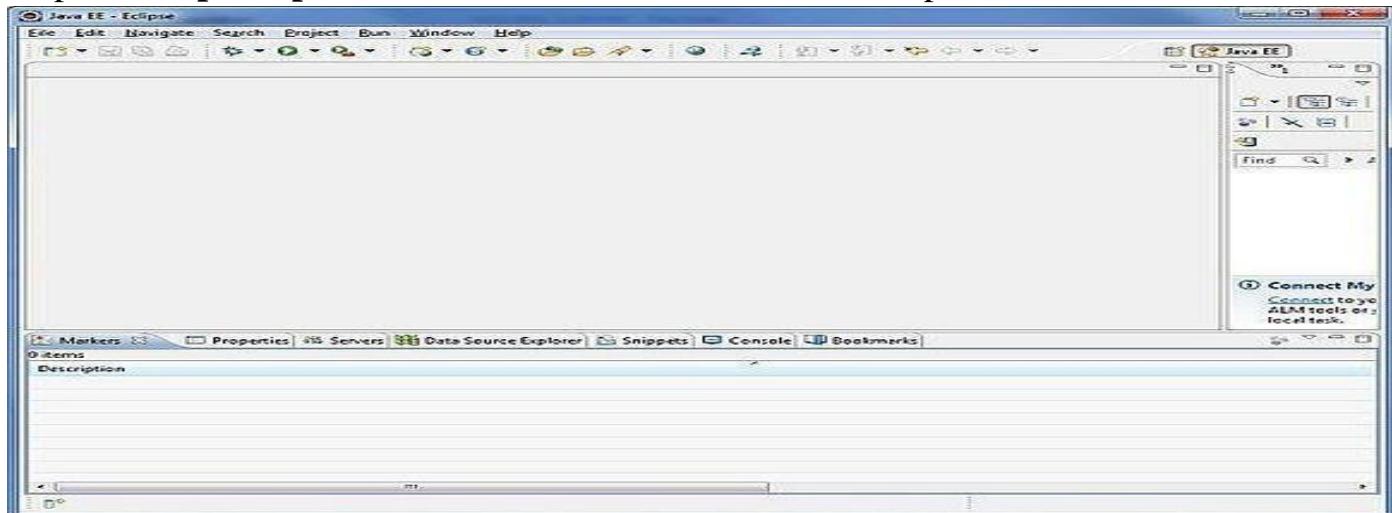
Finally, select the Create AVD button to see how to create the virtual device called demo AVD.

You now have everything ready for developing Android applications—the Android SDK, the Android platform, the Eclipse IDE, the ADT plug-in, and an AVD for testing Android applications. You can now create your first Android application.

1.7 CREATING THE FIRST ANDROID PROJECT

Now let's go over how to set up your first project so all you'll have left to do is write! you'll start a new Android Studio project and get to know the project workspace, including the project editor that you'll use to code the app.

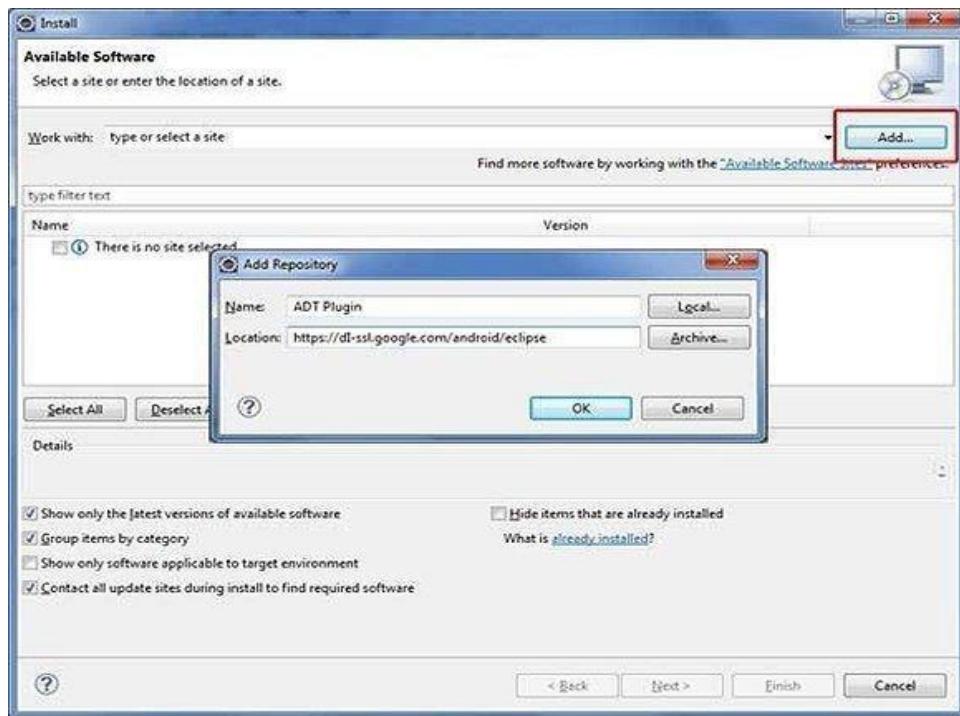
Step 1: Setup Eclipse IDE: Install the latest version of Eclipse. After successful installation, it



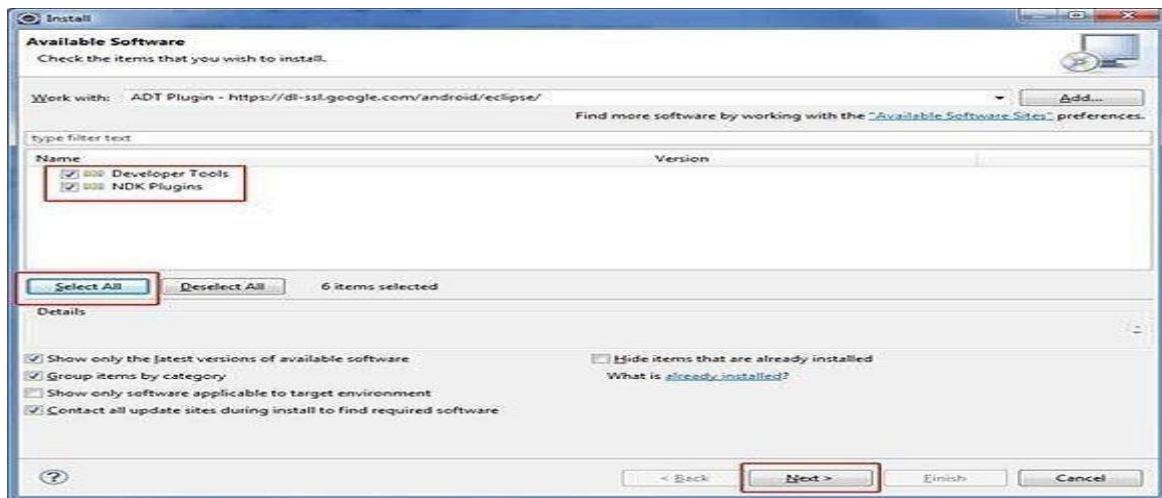
should display a window like above

Step 2: Setup Android Development Tools (ADT) Plug-in

Here you will learn to install the Android Development Tool plug-in for Eclipse. To do this, you have to click on **Help > Software Updates > Install New Software**. This will display the following dialogue box.



Just click on the Add button as shown in the picture and add <https://dl-ssl.google.com/android/eclipse/> as the location. When you press OK, Eclipse will start to search for the required plug-in and finally it will list the found plug-ins.



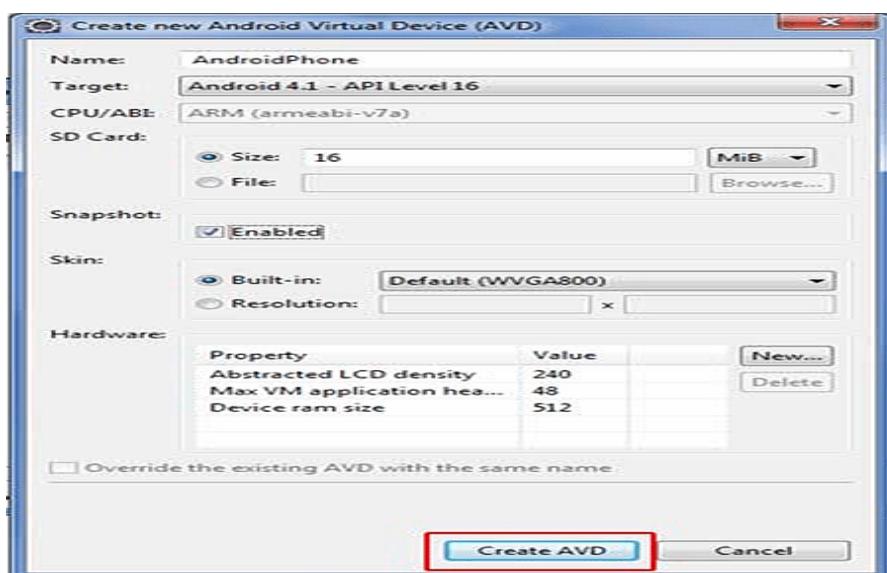
Step 3: Configuring the ADT plug-in

After the installing ADT plug-in, now tell the eclipse IDE for your android SDK location. To do so:

1. Select the **Window menu > preferences**
2. Now select the android from the left panel. Here you may see a dialog box asking if you want to send the statistics to the Google. Clicks proceed.
3. Click on the browse button and locate your SDK directory e.g. my SDK location is C:\Program Files\Android\android-sdk.
4. Click the apply button then OK.

Step 4: Create Android Virtual Device:

The last step is to create Android Virtual Device, which you will use to test your Android applications. To do this, open Eclipse and Launch Android AVD Manager from options **Window > AVD Manager** and click on **New** which will create a successful Android Virtual Device. Use the screenshot below to enter the correct values.



1.8 USING THE TEXTVIEW CONTROL

In android **ui** or **input** controls are the interactive or View components which are used to design the user interface of an application. In android we have a wide variety of UI or input controls available, those are TextView, EditText, Buttons, Checkbox, Progressbar, Spinners, etc.

In android, **TextView** is a user interface control which is used to set and display the text to the user based on our requirements. The TextView control will act as like label control and it won't allow users to edit the text. A good example of TextView control usage would be to display textual labels for other controls, like "Enter a Date:", "Enter a Name:" or "Enter a

Password:".

In android, we can create a TextView control in two ways either in XML layout file or create it in Activity file programmatically.

Specific attributes of TextView controls you will want to be aware of:

- Give the TextView control a unique name using the id property.
- Set the text displayed within the TextView control using the text property; programmatically set with the setText() method.
- Set the layout height and layout width properties of the control as appropriate.
- Set any other attributes you desire to adjust the control's appearance. For example, adjust the text size, color, font or other style settings.
- By default, text contents of a TextView control are left-aligned. However, you can position the text using the gravity attribute. This setting positions your text relative to the control's overall width and height and only really makes sense to use if there is whitespace within the TextView control.
- In XML, this property would appear within your TextView control as:

android:gravity="center"

- By default, the background of a TextView control is transparent. That is, whatever is behind the control is shown. However, you can set the background of a control explicitly, to a color resource, or a drawable (picture). In XML, this property would appear within your TextView control as:

android:background="#0000ff"

- By default, any text contents within a TextView control is displayed as plain text. However, by setting one simple attribute called autoLink, all you can enable automatic detection of web, email, phone and address information within the text. In XML, this property would appear within your TextView control as:

android:autoLink="all"

- You can control the color of the TextView control text within the by using the textColor attribute. This attribute can be set to a color resource, or a specific color by hex value. In XML, this property would appear within your TextView control as:

android:textColor="#ff0000"

- You can control the style of the text (bold, italic) and font family (sans, serif, monospace) within the TextView control by using the textStyle and typeface attributes. In XML, these properties would appear within your TextView control as:

android:textStyle="bold"

android:typeface="monospace"

Example:

```

<TextView android:id="@+id/message"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context=".HelloWorldAppActivity"
    android:typeface="serif"
    android:textColor="#0F0"
    android:textSize="25dp"
    android:textStyle="italic"
    android:gravity="center_horizontal" />

```

This code makes the text of the TextView control appear in serif font, green color, 25dp size, italic, and at the horizontal center of the container

1.9 USING THE ANDROID EMULATOR

The Android emulator is used for testing and debugging applications before they are loaded onto a real handset. Android emulator is typically used for deploying apps that are developed in your IDE without actually installing it in a device. Android emulators such as Blue stacks can run android apps where in which emulators like AVD and give motion are used to emulate an entire operating system. The Android emulator is integrated into Eclipse through the ADT plug-in.



Limitations of the Android Emulator

The Android emulator is useful to test Android applications for compatibility with devices of different configurations. But still, it is a piece of software and not an actual device and has several limitations:

- Emulators no doubt help in knowing how an application may operate within a given environment, but they still don't provide the actual environment to an application. For example, an actual device has memory, CPU, or other physical limitations that an emulator

doesn't reveal.

- Emulators just simulate certain handset behavior. Features such as GPS, sensors, battery, power settings, and network connectivity can be easily simulated on a computer.
- SMS messages are also simulated and do not use a real network.
- Phone calls cannot be placed or received but are simulated.
- No support for device-attached headphones is available.
- Peripherals such as camera/video capture are not fully functional.
- No USB or Bluetooth support is available.

The emulator provides some facilities too. You can use the mouse and keyboard to interact with the emulator when it is running. For example, you can use your computer mouse to click, scroll, and drag items on the emulator. You can also use it to simulate finger touch on the soft keyboard or a physical emulator keyboard. You can use your computer keyboard to input text into UI controls and to execute specific emulator commands. Some of the most commonly used commands are

- Back [ESC button]
- Call [F3]
- End [F4]
- Volume Up [KEYPAD_PLUS, Ctrl-5]
- Volume down [KEYPAD_MINUS, Ctrl-F6]
- Switching orientations [KEYPAD_7, Ctrl-F11/KEYPAD_9, Ctrl-F12]

You can also interact with an emulator from within the DDMS tool. Eclipse IDE provides three perspectives to work with: *Java perspective*, *Debug perspective*, and *DDMS perspective*. The Java perspective is the default and the one with which you have been working up to now. You can switch between perspectives by choosing the appropriate icon in the top-right corner of the Eclipse environment. The three perspectives are as follows:

- **The Java perspective**—It's the default perspective in Eclipse where you spend most of the time. It shows the panes where you can write code and navigate around the project.
- **The Debug perspective**—Enables application debugging. You can set breakpoints; step through the code; view LogCat logging information, threads, and so on.
- **The Dalvik Debug Monitor Service (DDMS) perspective**—Enables you to monitor and manipulate emulator and device status. It also provides screen capture and simulates incoming phone calls, SMS sending, and GPS coordinates. To manage content in the device or emulator, you can use the ADB (Android Debug Bridge).

1.10 THE ANDROID DEBUG BRIDGE (ADB)

The Android-Debug-Bridge (abbreviated as adb) is a software-interface for the android system, which can be used to connect an android device with a computer using an USB cable or a wireless connection. It can be used to execute commands on the phone or transfer data between the device and the computer.

The tool is part of the Android SDK (Android Software Development Kit) and is located in the subdirectory platform-tools. In previous versions of the SDK it was located in the subdirectory tools.

The Android Debug Bridge is a software interface between the device and the local computer, which allows the direct communication of both components. This includes the possibility to transfer files from one component to the other one, as well as executing commands from the computer on the connected device. The ADB can be used through a command line windows, terminal/shell in Linux-based systems, a command line (cmd) for Windows. The main advantage is to execute commands on the phone directly out of the computer, without any direct user interaction to the phone, which makes especially debugging a lot easier.

It is a client-server program that includes three components:

- **A client**, which sends commands. The client runs on your development MACHine. You can invoke a client from a command-line terminal by issuing an adb command.
- **A daemon (adb)**, which runs commands on a device. The daemon runs as a background process on each device.
- **A server**, which manages communication between the client and the daemon. The server runs as a background process on your development MACHine.

When you start an adb client, the client first checks whether there is an adb server process already running. If there isn't, it starts the server process. When the server starts, it binds to local TCP port 5037 and listens for commands sent from adb clients—all adb clients use port 5037 to communicate with the adb server.

The server then sets up connections to all running devices. It locates emulators by scanning odd-numbered ports in the range 5555 to 5585, the range used by the first 16 emulators. Where the server finds an adb daemon (adb), it sets up a connection to that port. Note that each emulator uses a pair of sequential ports — an even-numbered port for console connections and an odd-numbered port for adb connections.

Once the server has set up connections to all devices, you can use adb commands to access those devices. Because the server manages connections to devices and handles commands from multiple adb clients, you can control any device from any client

1.11 LAUNCHING ANDROID APPLICATIONS ON A HANDSET

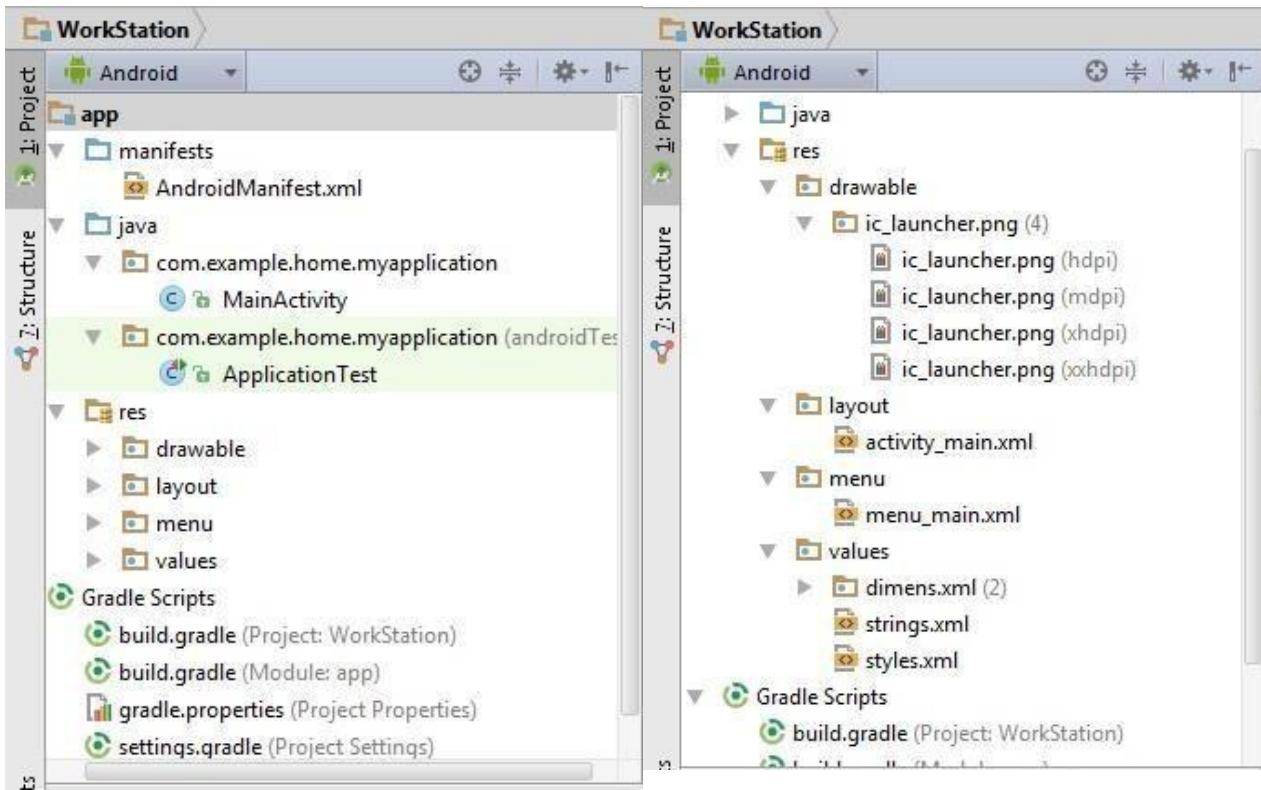
To load an application onto a real handset, you need to plug a handset into your computer, using the USB data cable. You first confirm whether the configurations for debugging your application are correct and then launch the application as described here:

- 1.**In Eclipse, choose the Run, Debug Configurations option.
- 2.**Select the configuration HelloWorldApp_configuration, which you created for the HelloWorldApp application.
- 3.**Select the Target tab, set the Deployment Target Selection Mode to Manual. The Manual option allows us to choose the device or AVD to connect to when using this launch configuration.
- 4.**Apply the changes to the configuration file by clicking the Apply button.
- 5.**Plug an Android device into your computer, using a USB cable.
- 6.**Select Run, Debug in Eclipse or press the F11 key. A dialog box appears, showing all available configurations for running and debugging your application. The physical device(s) connected to the computer are also listed. Double- click the running Android device. Eclipse now installs the Android application on the handset, attaches a debugger, and runs the application.

Android Studio Project Structure

Android studio shows a structured view of the project files and also provides a quick access to source files also. Android studio groups all the required source files, build files, resource files of all the modules at the top of project view.

Let's first understand the anatomy of the Android Studio:



1).**idea:** This folder contains the directories (subfolders) for Interlay IDEA settings.

2) **app:** It contains the actual application code(source files, resource file and manifest files). It further contains the following sub-directories :

a) **bulid:** This folder has sub-folders for the build-variants. The app/build/output/apk directory contains packages named app-<flavor>-<built-type>.apk. So different variant of the single app resides here.

b) **libs:** As the name suggests, this folder has all the .jar files and the library files.

3) **src:** Here you will see two sub-directories androidTest and main.

In **androidTest** the application code for testing purpose is created by android automatically.

This helps in building testing packages without modifying the building files and the application code.

main contains the all the source .java files including the stub mainactivity.java.

The **res** directory is where you will find the resources like drawable files, menu, values (style files, string values, dimension values).

Sub-directories of res folder

- **anim/**: For XML files that are compiled into animation objects.

- color/: For XML files that describe colors.
- drawable/: For image files (PNG, JPEG, or GIF), 9-Patch image files, and XML files that describe Drawable shapes or Drawable objects that contain multiple states (normal, pressed, or focused).
- mipmap/: For app launcher icons. This folder contains additional sub-folders for different screen resolutions. Once an image is imported into this folder, Studio automatically resizes the image into different resolutions and places them in the appropriate folders. This behavior allows launcher apps to pick the best resolution icon for your app to display on the home screen. To see how to import an image into the mipmap folder, please read about using 'Image asset'.
- layout/: XML files that are compiled into screen layouts (or parts of a parent layout).
- menu/: For XML files that define application menus.
- values/: For XML files that define constants that can be accessed in other XML and Java files.

R.java

R.java is an un-editable file auto-generated by Studio, whenever we modify the XML content. This file links the XML values to Java files. Whenever we need to use XML values in a Java file, we call these values using the R class.

The following images show a sample XML file, where a string variable named 'title' is assigned a value of 'Internshala - Tic Tac Toe'. This variable is then accessed in a java file using the R class.

4)Gradle scripts: With Android studio, Google switched to the new advanced building system, Gradle. It is a JVM based build system. If you want to make the package building task automatically, then you can write your own script in java or groovy and distribute it. Gradle allows to create different variants apk files for the same application project. We will learn about Gradles in the later chapters.

5)External Libraries: This is the place where all the referenced libraries and the information about the targeted platform SDK are stored.

What is Build system and Gradle

The build system is responsible to build, test an android system and also prepare the deployable files for the specified platform. In simple words, build system generates the .apk files for the application project.

With Android Studio, the advanced build system allows the developers to configure the build systems manually, create different variant APK files from single project

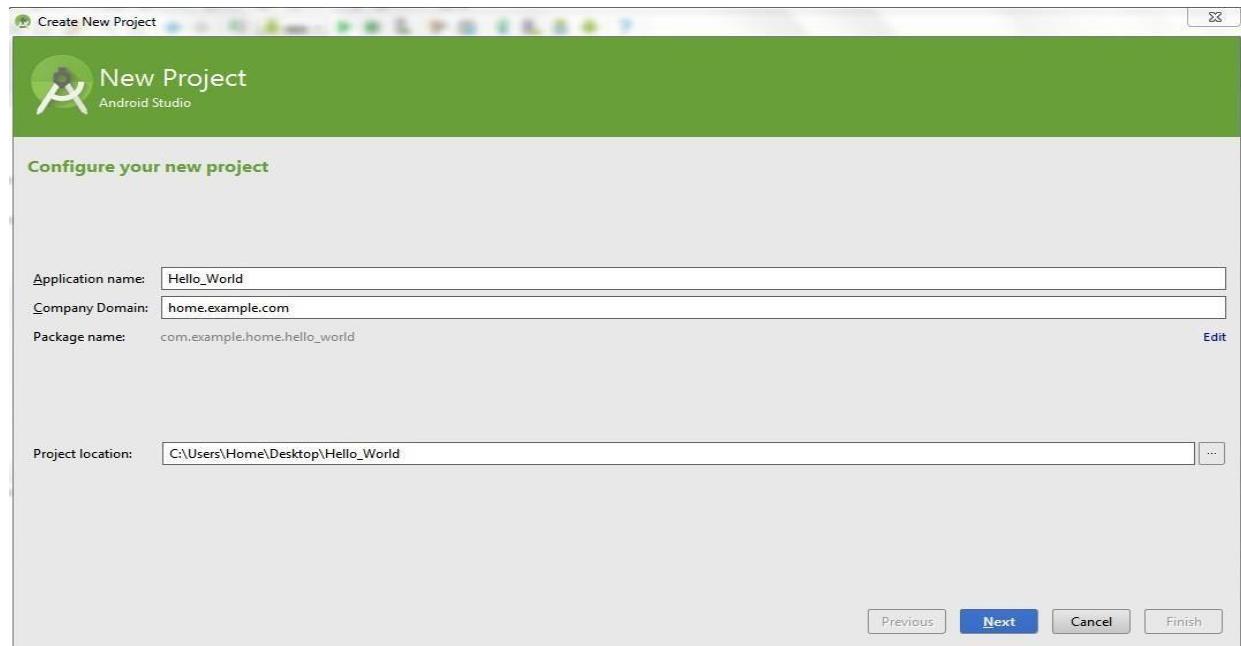
(without modifying the execution code) and share the code and resources from other modules. Before Android Studio, Eclipse was the IDE used for android development. Eclipse kept all the .java files (in src directory) and resource files (in res directory) in the same directory. That allows the build system to group all the files into an intermediate code and finally generates .apk file.

Android Studio uses Gradle as its build system. One intriguing feature that Gradle offers is that it allows you to write your own script to automate the task of building an app. As Gradle is plug-in based system, if you have your own programming language then you can write the plug-in in the script using java or groovy and share it with other developers too. Isn't that cool?

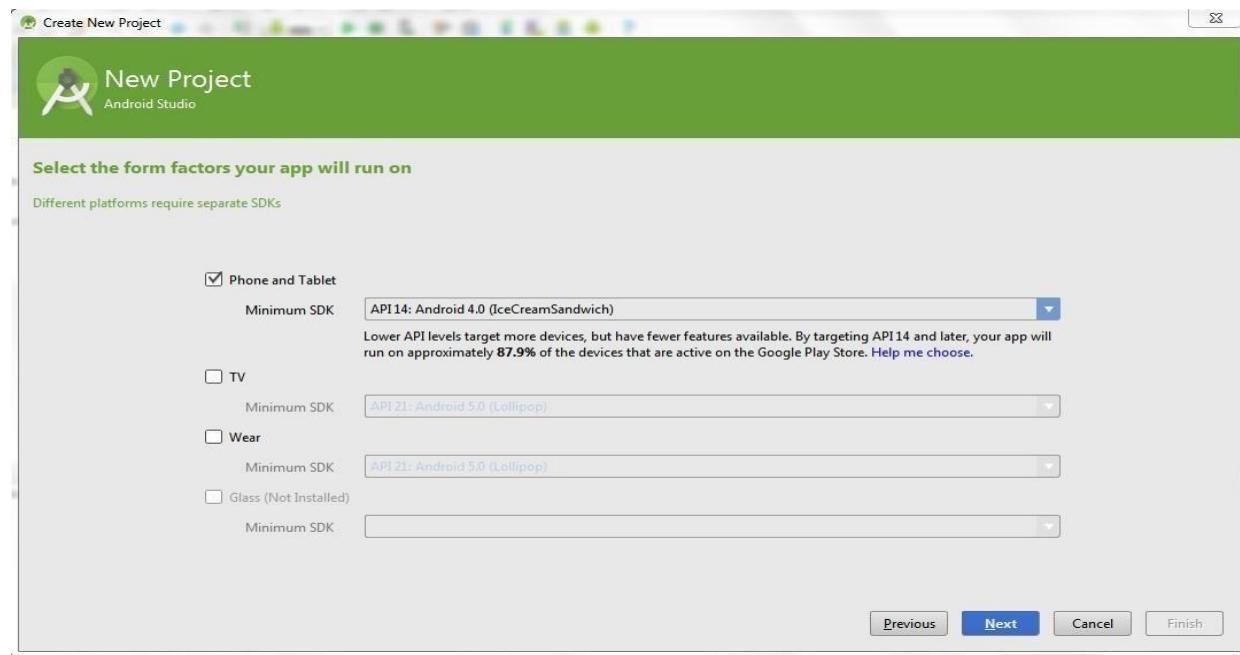
1.12 CREATING AN ARDROID PROJECT

'Hello World' APP

The first you will be creating will be "Hello World"! We will see how to start a project in Android Studio. First of all, launch Android Studio. Goto File-> New Project. Give the name of the application. Here we name it "Hello_World". Hit next.



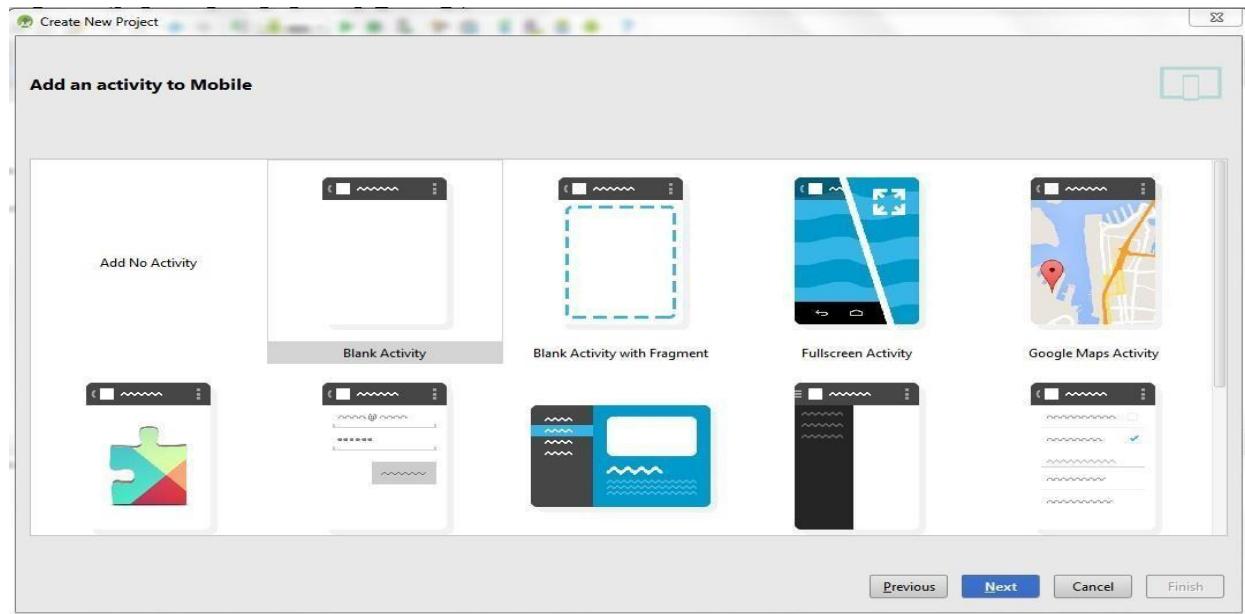
Select the category of the target devices and the desire minimum target platform of android:



Now, select the main activity style depending upon the needs. Here for “Hello_world”, select blank activity.

Name the main_activity files, which are the first activity to be displayed in the application. Click ‘finish’. Run the Application.

Building and Running an App



First you need to build your application project before running it on the device. Click on the “build” icon from the toolbar and then select “Make project”.

To Run an application:

Select “Run” from the toolbar and then “run app”.

You can run your app on the emulator or on a real-device from Android Studio. This is done using the debug version of the app. Run configuration defines which module will run, which activity is start, and about all the AVD settings. If you run the android application for the first time, android studio will automatically create a run configuration and choose AVD to run it. Though you can create or modify the run configuration.

Run an app on Emulator:

When you run an app on the emulator, first make sure about the AVD (Android Virtual Device). You can choose from the available AVD or create a new AVD.

Go to Tools >Android > AVD Manager. Click on ‘create virtual device’.

Choose Hardware category and configuration, Click next. Select the desired system version and create the AVD.

To run application on the desired AVD, click on the launch button.

Run an App on the real-device :

First, you have to make sure that the real-device in which you desire to run your app, is debuggable.

1)Check if the android:debuggable attribute is set to true in the build.gradle file. If not then, you cannot debug it. Make it true in case you want to run it on real-device.

2) Please enable the USB Debugging, in the device. In android 4.2 or above, the developer option can be enabled by going to Settings > About Phone and tapping Build Number 7 times. Now you can see the Developer Option in the previous screen.

Now, when you have your AVD set up or the real-devices ready, go to Run > Run (OR Run > Debug). If you don’t see your device in the AVD window, then you need to download appropriate device drivers for your device from the internet.

CHAPTER 2

COMPONENTS OF ANDROID

2.1. UNDERSTANDING THE ROLE OF ANDROID APPLICATION COMPONENTS

Almost all popular applications are interactive. These applications interact with the user, and, depending on the data supplied by the user, desired actions and/or processing are performed. The user interface controls thus play a major role in getting feedback from the user.

Application components are the ones which when combined together, offers you a brilliant Android application. So, these components exactly act as the building blocks of an Android application. The information regarding all the application components is provided in the manifest file, which is **AndroidManifest.xml**. This file will help you in understanding the use of each and every application component and how do they interact with each other. Android provides four important components to build any android application.

- Activities
- Services
- Intent and broadcast receivers
- Content Providers



Figure Principal Ingredients of android application

1. Activities-

Activities are said to be the presentation layer of our applications. An activity is the first stepping stone in building an Android user application. The UI of our application is built around one or more extensions of the Activity class.

An activity in android is like your computer welcome screen which presents single user display. In other words, Activity in android represents single screen with a user interface. We can understand Activity in terms of web applications. For example: We create numbers of web pages to build complete web application, similarly on the other hand android application consist of several Activities to run as a complete application. There is one “main” activity. All other activities are child activities. There is a stack called back stack. Whenever, there is a new window

is started, previous activity is pushed to the back stack and it is stopped until the new activity is done. As soon as the back key of your device is pressed, new activity is popped out of stack and destroyed. Now previous activity resumes.

2. Services-

These are like invisible workers of our app. These components run at backend, updating your data sources and Activities, triggering Notification and also broadcast Intents. They also perform some tasks when applications are not active. This component is responsible for handling the time taking operations which generally runs in the background of the operating system (in this case, Android). The simple example of the service component is that when you play music on your mobile phone, you will be able to use other applications too. A service can take two forms:

1. **Started:** After a service starts, it can run indefinitely and usually performs single operation. No result is returned to user. For example, uploading a file. After the task is completed, it should terminate itself.
2. **Bound:** In this case, a component is bound to a service so that a particular task can be completed. This type of service provides a client-server like interface. Requests can be send, receive requests, and return result to the user. Inter process communication is achieved through this service.

3. Intents and Broadcast Receivers-

Android Intent is the *message* that is passed between components such as activities, content providers, broadcast receivers, services etc. It is generally used with `startActivity()` method to invoke activity, broadcast receivers etc. It binds individual components to each other.

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action. another example is, when your device boots up or switched on, the system generates a broadcast to all apps. There should be a procedure or should be something which can receive these broadcasts. These receptors are called broadcast receivers. There are two types of broadcasts:

1. **Normal Broadcasts:** These are asynchronous in nature. Many receivers can be activated at the same time which doesn't have any defined order. But they are very efficient.
2. **Ordered Broadcasts:** They are synchronous in nature. Broadcast received by one receiver passes it to other receivers. Broadcasts are delivered to receiver on one-to-one and sequential basis. Either receiver will pass result to another receiver or it may completely destroy the broadcast.

Content Providers-

It is used to manage and persist the application data also typically interact with SQL database. They are also responsible for sharing the data beyond the application boundaries. The Content

Providers of a particular application can be configured to allow access from other applications, and the Content Providers exposed by other applications can also be configured.

With content providers we can save data in SQLite database, on the web or any other persistent storage location, where application can easily access the data. This component is useful in reading and writing private data. For example: we can read and write important reminders or notes in database (within an application).

Android Widgets and Notifications

Android App widgets are the small application views. These views can be embedded into other applications. They can receive updates on periodic basis. A widget is a quick view of your app's functionality and data. This view is accessible from home screen of your device. Now widgets are of following types:

1. Informational Widget: These Android widgets are going to display only that information to user which is important and dynamic in nature. Example the information displayed on your home screen saying time and weather condition is a widget of this type.

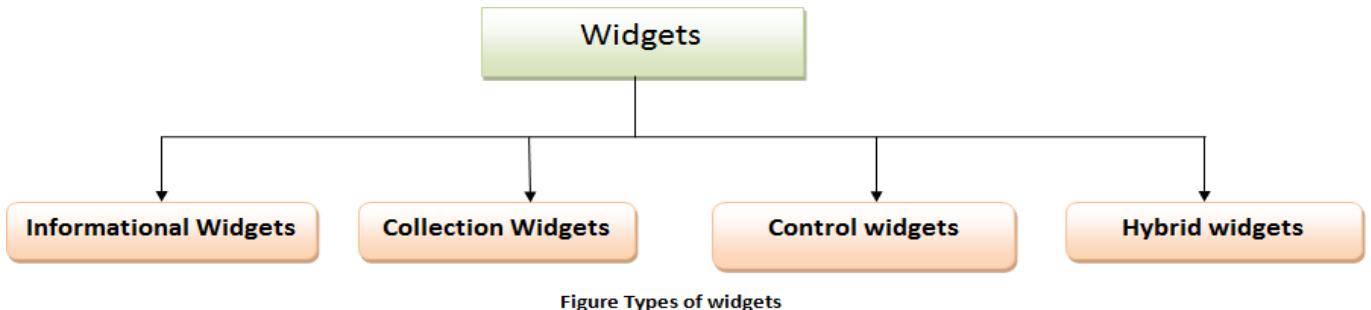


Figure Types of widgets

2. Collection Widgets: These Android widgets scroll in top-to-down direction. Collection of information of same type and then enabling user to open any one of them to full detail. Example is your e-mail app which will display all the mails in your inbox and then allow you to open any one of them.

3. Control Widgets: Displays the most frequently used functionalities which user might want to control from home screen. For example in a video app, you can pause, play, stop, go to previous track, move to next track is an example of control widget.

4. Hybrid Widgets: These Android widgets combine features of all of the above three.

Notification, as the name says keeps the user aware of events going on. User is kept informed like any news channel. For e.g, everyone of us know about facebook or whatsapp, now notification system of app is responsible for informing you about any new friend request, chat request, or a new message from say, dvs or xyz, etc.

There are a few other application components that you should be aware of. These application components include fragments, views, layouts, intents, resources, and manifest. All of these components are used for the creation of above components.

S.No.	Application Components	Description
1	Fragments	* Represents the fragments of a user interface in the Activity component
2	Views	* Includes the user interface elements like buttons, drop-down lists, etc.
3	Layouts	* Controls the screen format based on different hierarchies of the views * Takes care of the appearance of the views on the screen
4	Intents	* Wires the messages of different components together
5	Resources	* Includes external elements like drawable or editable pictures, strings, and constants
6	Manifest	* Carries the information regarding the applications * Configuration file

2.2. UNDERSTANDING THE UTILITY OF ANDROID API

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an application, you're using an API.

When you use an application on your mobile phone, the application connects to the Internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way. This is what an API is - all of this happens via API.

In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools for building an application. An API specifies how software components should interact and APIs are used when programming graphical user interface (GUI) components. In general terms, it is a set of clearly defined methods of communication between various components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

The Android platform provides a framework API that applications can use to interact with the underlying Android system. The framework API consists of:

- A core set of packages and classes
- A set of XML elements and attributes for declaring a manifest file
- A set of XML elements and attributes for declaring and accessing resources
- A set of Intents
- A set of permissions that applications can request, as well as permission enforcements included in the system

The following table specifies the API Level supported by each version of the Android platform.

Codename	Version	API level/NDK release
Android11	11	API level 30
Android10	10	API level 29
Pie	9	API level 28
Oreo	8.1.0	API level 27
Oreo	8.0.0	API level 26
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23
Lollipop	5.1	API level 22
Lollipop	5.0	API level 21
KitKat	4.4 - 4.4.4	API level 19
Jelly Bean	4.3.x	API level 18
Jelly Bean	4.2.x	API level 17
Jelly Bean	4.1.x	API level 16
Ice Cream Sandwich	4.0.3 - 4.0.4	API level 15, NDK 8
Ice Cream Sandwich	4.0.1 - 4.0.2	API level 14, NDK 7
Honeycomb	3.2.x	API level 13
Honeycomb	3.1	API level 12, NDK 6
Honeycomb	3.0	API level 11
Gingerbread	2.3.3 - 2.3.7	API level 10
Gingerbread	2.3 - 2.3.2	API level 9, NDK 5
Froyo	2.2.x	API level 8, NDK 4
Eclair	2.1	API level 7, NDK 3
Eclair	2.0.1	API level 6
Eclair	2.0	API level 5
Donut	1.6	API level 4, NDK 2
Cupcake	1.5	API level 3, NDK 1
(no codename)	1.1	API level 2
(no codename)	1.0	API level 1

Each successive version of the Android platform can include updates to the Android application framework API that it delivers. The framework API is specified through an integer called API level, and each Android platform version supports exactly one API level, although

backward compatibility is there; that is, earlier API levels are supported. The initial release of the Android platform provided API Level 1, and subsequent releases have incremented the API level.

Updates to the framework API are designed so that the new API remains compatible with earlier versions of the API. That is, most changes in the API are additive and introduce new or replacement functionality. As parts of the API are upgraded, the older replaced parts are deprecated but are not removed, so that existing applications can still use them. In a very small number of cases, parts of the API may be modified or removed.

The framework API that an Android platform delivers is specified using an integer identifier called "API Level". Each Android platform version supports exactly one API Level, although support is implicit for all earlier API Levels (down to API Level 1). The initial release of the Android platform provided API Level 1 and subsequent releases have incremented the API Level.

Uses of API Level in Android

The API Level identifier serves a key role in ensuring the best possible experience for users and application developers:

- It lets the Android platform describe the maximum framework API revision that it supports
- It lets applications describe the framework API revision that they require
- It lets the system negotiate the installation of applications on the user's device, such that version- incompatible applications are not installed.
- Each Android platform version stores its API Level identifier internally, in the Android system itself.

2.3. OVERVIEW OF THE ANDROID PROJECT FILES

The following files and directories are created for the Android application. The list below is just an overview of the files and directories

• **/src folder**— The folder that contains the entire Java source file of the application. The folder contains a directory structure corresponding to the package name supplied in the application. The folder contains the project's default package: com.company.basicelloworld. On expanding the package, you find the Activity of the application, the MainActivity.java file, within it.

• **/src/com.company.basicelloworld** — Refers to the package name of the application. To avoid any collision among the class names, variable names, and so on used in the application with those of other Android applications, each application has to be packaged in a unique container.

• **/src/com.company.helloworld/MainActivity.java**- The default Activity file of the application. Recall that each application has at least one Activity that acts as the main entry point to the application. The Activity file is automatically defined as the default launch Activity in the Android Manifest file.

• **/gen folder**—Contains Java files generated by ADT on compiling the application. That is, the

gen folder will come into existence after compiling the application for the first time. The folder contains two files

1. **R.java:** file that contains references for all the resources defined in the res directory.
2. **BuildConfig.java:** file that is used to run code only in debug mode. It contains a DEBUG constant that helps in running debug-only functions.

• **/gen/com.company.basicelloworld/R.java**—All the layout and other resource information that is coded in the XML files is converted into Java source code and placed in the R.java file. It also means that the file contains the ID of all the resources of the application. The R.java file is compiled into the Java byte code files and then converted into .dex format. You should never edit this file by hand, as it is automatically overwritten when you add or edit resources.

• **/assets folder**—The assets folder is empty by default. It stores raw asset files that may be required in the application. It may contain fonts, external JAR files, and so on to be used in the application. The assets folder is like a resource folder where un-compiled resources of the project are kept and no IDs are generated for the resources kept here.

- **Android SDK jar file**—The jar file for the target platform
- **/bin folder**—The folder that stores the compiled version of the application.
- **/res folder**—The folder where all application resources (images, layout files, and string files) are kept. Instead of hard coding an image or string in an application, a better approach is to create a respective resource in the res folder and include its reference in the application. Each resource is assigned a unique resource ID, which automatically appears in the R.java file and thus in the R class after compilation, enabling us to refer to the resource in the code
- **/res/drawable-xhdpi, /res/drawable-hdpi, /res/drawable-mdpi, /res/drawable-ldpi**—the application's icon and graphic resources are kept in these folders. Because devices have screens of different densities, the graphics of different resolutions are kept in these folders. Usually, graphics of 320dpi, 240dpi, 160dpi, and 120dpi are used in Android applications. The application picks up the graphic from the correct folder after determining the density of the current device.
- **/res/layout**—Stores the layout file(s) in XML format.
- **/res/values**—Stores all the values resources. The values resources include many types such as string resource, dimension resource, and color resource.
- **/res/layout/activity_main.xml**—The layout file used by MainActivity to draw views on the screen. The views or controls are laid in the specified layout.
- **/res/values/strings.xml**—Contains the string resources. String resources contain the text matter to be assigned to different controls of the applications.
- **AndroidManifest.xml**—The central configuration file for the application. It is one of the most important file in the Android project structure. It contains information of the package, including components of the application such as activities, services, broadcast receivers, content

providers.

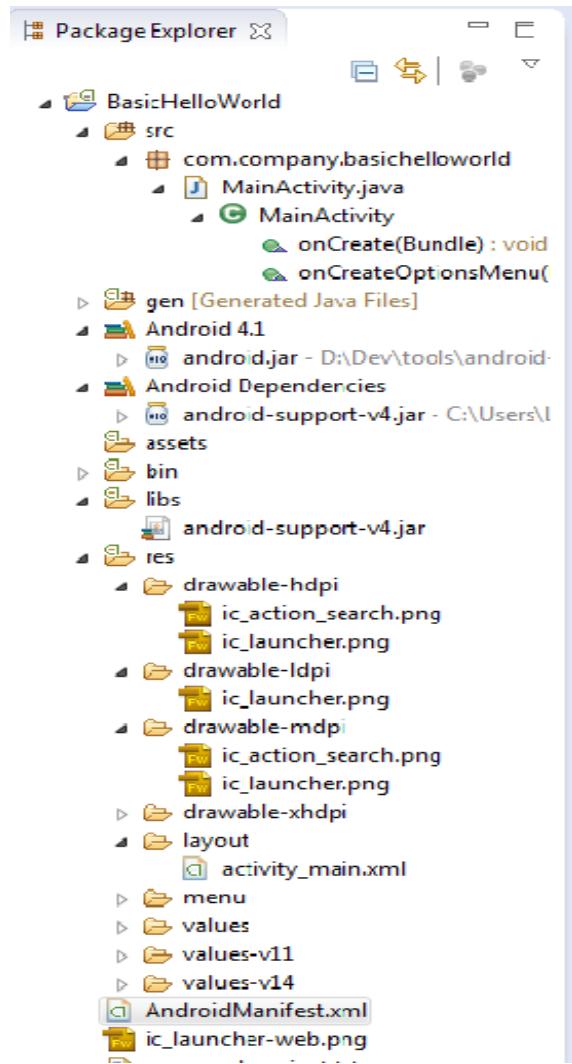


Figure: Package Explorer window showing different directories, subdirectories, and files automatically created by ADT

- **project.properties**—A build file used by Eclipse and the Android ADT plug-in. It contains project settings such as the build target. You can use this file to change various properties of the project. If required, the file should not be edited directly but through editors available in Eclipse.

It's clear that the Java Activity file is the main file that drives the entire Android application.

2.4 UNDERSTANDING ACTIVITIES

An Activity is a single screen in Android. It is like a window in a desktop app, or a Frame in a Java program. An Activity allows you place all your UI components or widgets together on the screen. Every unique screen the user interacts with in an application is displayed through an Activity—one Activity for each screen. Users can interact with an application by performing

different actions with the visual controls found in the Activity. A simple application may consist of just one Activity, whereas large applications contain several Activities. Each Activity of an application operates independently of the others.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

A stack of Activities is maintained while running an application, and the Activity at the top of the stack is the one currently being displayed. When the Back button is pressed, the Activity is popped from the stack, making the previous Activity the current Activity, which therefore displays the previous screen. The transition from one Activity to another is accomplished through the use of asynchronous messages called intents. Intents can be used to pass data from one Activity to another. All of the Activities in the application must be defined in the Application's manifest file, an XML file that keeps track of all the components and permissions of the application.

Each Activity in an Android application is either a direct subclass of the Activity base class or a subclass of an Activity subclass.

Activity life cycle

Activities have life cycles and inherit a set of life cycle methods from the Activity base class that are executed when the corresponding life cycle state of the Activity is reached.

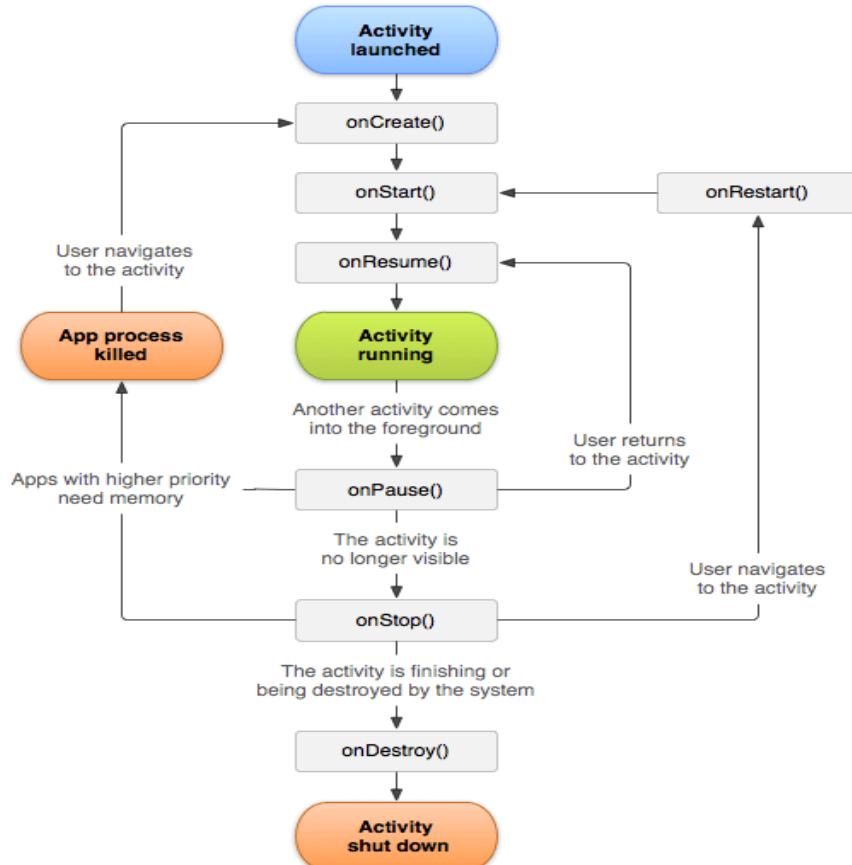
The Android Activity life cycle defines the states or events that an Activity goes through from the time it is created until it finishes running. The Activity monitors and reacts to these events by executing methods that override the Activity class methods for each event.

Each time the Activity state changes, one of the following lifecycle methods will be called on the Activity class. The lifecycle method of Activity describes how activity will behave at different states

- **onCreate():** This is called when the Activity is first initialized. You need to implement this method in order to do any initialization specific to your Activity. Views are created, bind data, etc. A Bundle object is passed where all the previous activity states are captured.
- **onStart():** This is called the first time that the Activity is about to become visible to the user, as the Activity prepares to come to the foreground become interactive. Once this callback finishes, the onResume() method will be called.
- **onResume():** When the Activity goes into this state, it begins to interacts with the user. The Activity continues in this state till something happen to take focus from the app or Activity (such as an incoming call). When this happens, the onPause() method will be

called.

- **onPause():** This method is used to pause operations that should not happen when the Activity is in paused state. A call to this method indicates that the user is leaving the app. For example, in a music player app, an incoming call will cause the app to transition into a paused state. This should mute or pause the currently playing music. When the user returns to the app, the onResume() method will be called.
- **onStop():** This method is called when the Activity is no longer visible in the app. It can happen, for example, when another Activity has been loaded and is taking the full screen of the device. When this method is called, the Activity is said to be in a stopped state. In this state, the system either calls the onRestart() to bring back interactivity with Activity. Or it calls the onDestroy() method to destroy the Activity.
- **onDestroy():** This gets called before the Activity is destroyed. The system calls this method when a user terminates the Activity, or because the system is temporarily destroying the process that contains the Activity to save space. Be sure to free up any resources your Activity has created in this method, or else your app will have a memory leak!



- **onRestart():** This gets called when an Activity restarts after it had been stopped.

All the activities of an application, permissions, and intents are defined through the XML-

structured manifest file AndroidManifest.xml. For each Activity, there needs to be an entry in the AndroidManifest.xml file, where you can define labels, permissions, and so on. In fact, the manifest file is the configuration file where all the different components of the application are defined.

Example: program to demonstrate the life cycle methods of an activity

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends
    Activity { @Override
protected void onCreate(Bundle
    savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d("lifecycle","onCreate invoked");
}
@Override
protected void onStart() {
    super.onStart();
    Log.d("lifecycle","onStart invoked");
}
@Override
protected void onResume() {
    super.onResume();
    Log.d("lifecycle","onResume invoked");
}
@Override
protected void onPause() {
    super.onPause();
    Log.d("lifecycle","onPause invoked");
}
@Override
protected void onStop() {
    super.onStop();
    Log.d("lifecycle","onStop invoked");
}
@Override
protected void onRestart() {
    super.onRestart();
    Log.d("lifecycle","onRestart invoked");
}
@Override
```

```

protected void onDestroy() {
    super.onDestroy();
    Log.d("lifecycle","onDestroy invoked");
}
}

```

2.5. ROLE OF THE ANDROID MANIFEST FILE

The AndroidManifest.xml is a configuration file. It is created by ADT when creating a new Android project and is kept in the project's root directory. The manifest file presents essential information about your app to the Android system, information the system must have before it can run any of the app's code.

It's an XML file that defines the overall structure and information about the application for example, the activities, services, and intents used in the application. It also contains the permissions that the application might need to run. The manifest also defines metadata of the application such as its icon and label. Besides this, the file also contains the information required in building and packaging the application for installing and deploying it on an Android device or the emulator. To sum up, the application manifest file contains all the essential information required by the Android system to run the application.

The manifest essentially fulfills the following roles:

- It makes key information about the app available without having to read other files in the package or run the application.
- It defines which components of the application (such as activities and services) will be accessible externally to other apps, and how those components interact with other apps.
- It unambiguously declares which components and resources (icons, text strings, etc.) to use in which cases, in particular with relation to other apps.
- It lists the libraries that the application must be linked against.

For example, if your app can accept files shared from other apps, the manifest says *which* component will receive the files, and *what kind* of files it can receive.

The main advantage of this approach is, it puts all of the information in one place, which reduces the chance of mistakes or human error when a developer is making an app. It is also an extensible system, so other kinds of data can be added later without breaking existing apps. And being able to read one file to get all the app's metadata can help with performance.

Default Code in the Android Manifest File

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.welcomemsg"

```

```

    android:versionCode="1"
    android:versionName="1.0" >
<uses-sdk android:minSdkVersion="8"
    android:targetSdkVersion="15" />
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".WelcomeMsgActivity"
        android:label="@string/title_activity_welcome_msg" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>
```

The `<manifest>` tag, is the root element of this XML document and contains several attributes:

- **android**—Identifies the Android namespace used to provide several system attributes used within the application.
- **package**—Its value is set to the application’s Java package. The name of the application package acts as the unique identifier for the application in the Android device.
- **versionCode/versionName**—The `versionCode` attribute is used to define the current application version. The version is used internally to compare application versions. The `versionName` attribute is used to specify a version number that is displayed to users.
- **<uses-sdk>**—This tag is optional and is used to specify the maximum, minimum, and preferred API level required for the application to operate. Three attributes are used with this tag as follows:
 - **android:minSdkVersion**—Used to specify the minimum API level required for this application to run. The default value is “1.” For example, the following attribute says that the minimum API level required by the application is 15: `android:minSdkVersion="15"`
 - Hence, the application will run on API Level 15 and above, but will not run on API Level 14 or below.
 - **android:targetSdkVersion**—Used to specify the preferred API level on which the application is designed to run.
 - **android:maxSdkVersion**—Used to specify the maximum API level supportable by the application; that is, the application cannot run on an API level higher than the one specified in

this attribute. While installing the application, the Android system checks the <uses-sdk> attributes defined in the application's manifest files and compares it with its own internal API level. The application can be installed only if

- ✓ The value of the android:minSdkVersion attribute of the application must be less than or equal to the system's API level. If the android:minSdkVersion attribute is not declared, it is assumed that the application requires API Level 1.
- ✓ The value of the android:maxSdkVersion attribute of the application (if it is declared) must be equal to, or greater than, the system's API level.

• **<application> tag**—Nested within <manifest> is <application>, which is the parent of application control tags. The icon and label attributes of the application tag refer to icon and label resources that will be displayed in Android devices to represent the application. The term "@drawable/icon" refers to the image file icon.png in the res/drawablefolder, and "@string/app_name" refers to the control resource with ID app_name in the strings.xml file that is stored in the res/values folder of the application.

• **<activity> tag**—Nested within <application> is the <activity> tag, which describes an Activity component. This tag's name attribute identifies a class that is an Activity, WelcomeMsgActivity. This name begins with a period character to show that it's relative to the com.androidunleashed.welcomemsg package. That is, the WelcomeMsgActivity is relative to <manifest>'s package value, com.androidunleashed.welcomemsg. Remember that on creating the new Android project WelcomeMsg, an Activity named WelcomeMsgActivity was automatically created for us in the specified package com.androidunleashed.welcomemsg by the ADT.

• **<intent-filter>**—Nested within <activity> is the <intent-filter>. The intents are used to interact with the applications and services. By default, the intent specifies the action as MAIN and the category as LAUNCHER; that is, it makes this Activity available from the application launcher, allowing it to launch when the application starts. Basically, the <intent-filter> tag declares the capabilities of different components through the nested <action> and <category> tags.

Besides the preceding default tags used in the manifest file, the following are some of the most commonly used tags:

• **<service> tags**—Used for declaring services. Services refer to the processes that run in the background without a user interface. They perform required processing and handle events silently without user interaction.

• **<receiver> tags**—Used for declaring broadcast receivers. Broadcast receivers are used to listen and respond to broadcast announcements. Applications initiate broadcasts for the interested applications when some event occurs that requires attention; for example, essential information or confirmation is required from the user before taking some destructive action. An

application can have any number of broadcast receivers. A broadcast receiver responds by taking a specific action.

- **<provider> tags**—Used for declaring content providers. Content providers provide content, that is, data to applications. They help in handling, storing, and sharing data such as audio, images, video, and contact lists with other applications. Android ships with a number of content providers that the applications can use to access and share data with other applications.

- **<uses-permission> tags**—Used for declaring the permissions that the application needs.

CHAPTER 3

BASIC CONTROLS

3.1 CREATING THE USER INTERFACE

There are three approaches to creating user interfaces in Android.

1. You can create user interfaces entirely in Java code
2. Entirely in XML
3. Combining both methods (defining the user interface in XML and then referring and modifying it through Java code).

The third approach, the combined approach, is highly preferred.

The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects.

- A View is an interactive UI component (or widget or control), such as button and text field. It controls a *rectangular area* on the screen. It is responsible for drawing itself and handling events (such as clicking, entering texts). Android provides many ready-to-use Views such as TextView, EditText, Button, RadioButton, etc, in package android.widget. You can also create your custom View by extending android.view.View.
- A ViewGroup is an *invisible container* used to *layout* the View components. Android provides many ready-to-use ViewGroups such as LinearLayout, RelativeLayout, TableLayout and GridLayout in package android.widget. You can also create your custom ViewGroup by extending from android.view.ViewGroup.

Views and ViewGroups are organized in a single tree structure called view-tree. You can create a view-tree either using programming codes or describing it in a XML layout file.

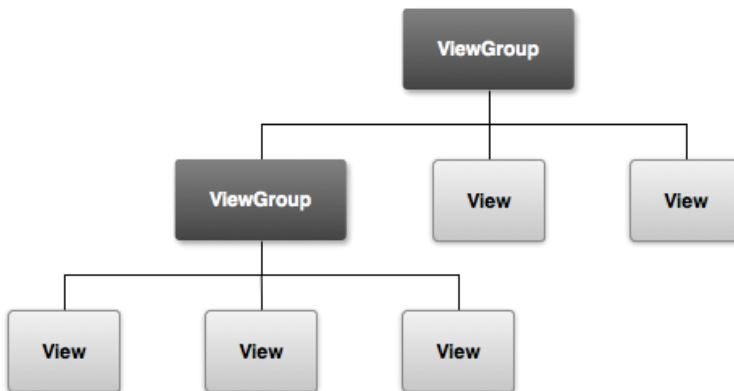


Figure: Illustration of how ViewGroup objects form branches in the layout and contain other View objects.

3.2 COMMONLY USED LAYOUTS AND CONTROLS

The views or the controls that we want to display in an application are arranged in an order or sequence by placing them in the desired layout. The layouts also known as Containers or ViewGroups. These are used for organizing the views or controls in the required format. Android provides the following layouts

- **LinearLayout:** In this layout, all elements are arranged in a descending column from top to bottom or left to right. Each element contains properties to specify how much of the screen space it will consume. Depending on the orientation parameter, elements are either arranged in row or column format.
- **RelativeLayout:** In this layout, each child element is laid out in relation to other child elements. That is, a child element appears in relation to the previous child. Also, the elements are laid out in relation to the parent.
- **AbsoluteLayout:** In this layout, each child is given a specific location within the bounds of the parent layout object. This layout is not suitable for devices with different screen sizes and hence is deprecated.
- **FrameLayout:** This is a layout used to display a single view. Views added to this are always placed at the top left of the layout. Any other view that is added to the FrameLayout overlaps the previous view; that is, each view stacks on top of the previous one.
- **TableLayout:** In this layout, the screen is assumed to be divided in table rows, and each of the child elements is arranged in a specific row and column.
- **GridLayout:** In this layout, child views are arranged in a grid format, that is, in the rows and columns pattern. The views can be placed at the specified row and column location. Also, more than one view can be placed at the given row and column position.

The following list highlights some of the controls commonly used in Android applications:

- **TextView:** A read-only text label. It supports multiline display, string formatting, and automatic word wrapping.
- **EditText:** An editable text box that also accepts multiline entry and word-wrapping.
- **ListView:** A ViewGroup that creates and manages a vertical list of views, displaying them as rows within the list.
- **Spinner:** A TextView and an associated list of items that allows us to select an item from the list to display in the text box.
- **Button:** A standard command button.
- **CheckBox:** A button allowing a user to select (check) or unselect (uncheck).
- **RadioButton:** A mutually exclusive button, which, when selected, unselects all other buttons in the group.

3.3 EVENT HANDLING

The action of clicking a Button, pressing the Enter key, or performing any action on any control is considered an **event**. The reaction to the event, that is, the action to be taken when the event occurs, is called **event handling**. To handle an event, you use the listeners that wait for an event occurrence. When an event occurs, the listeners detect it and direct the program to the appropriate routine.

An event listener is an interface in the View class that contains a single callback method, called an event occurrence. For example the callback method `onClick()` is called when the user clicks on a button. For event handling, the event listener is either implemented in the Activity class or is defined as an anonymous class. Thereafter, an instance of the implementation is passed to the respective control through the `setOnItemClickListener()` method.

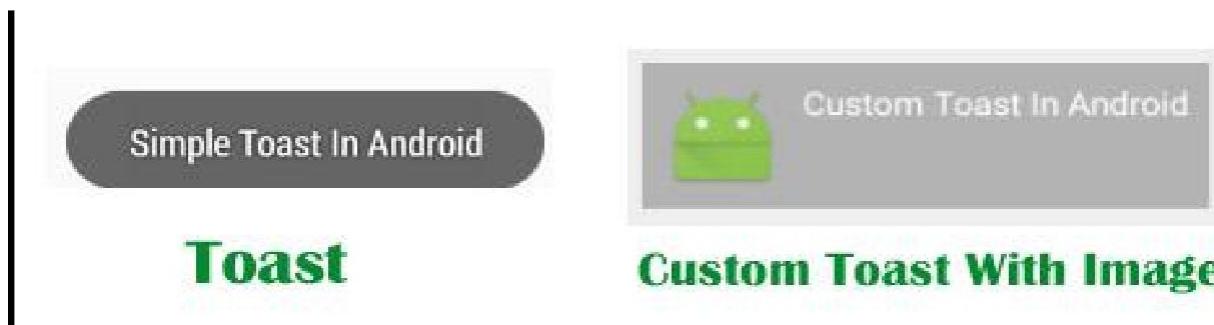
Note: Click is just one type of an event.

There are three ways of event handling:

- Creating an anonymous inner class
- Implementing the `OnItemClickListener` interface
- Declaring the event handler in the XML definition of the control

3.4 DISPLAYING MESSAGES THROUGH TOAST

In Android, Toast is used to display information for a period of time. It contains a message to be displayed quickly and disappears after specified period of time. It does not block the user interaction. Toast is a subclass of Object class. In this we use two constants for setting the



duration for the Toast. Toast notification in android always appears near the bottom of the screen. We can also create our custom toast by using custom layout(xml file).

Special Note: In Android, Toast is used when we required to notify user about an operation without expecting any user input. It displays a small popup for message and automatically fades out after timeout.

Important Methods Of Toast:

Let's we discuss some important methods of Toast that may be called in order to manage the Toast.

1. makeText (Context context, CharSequence text, int duration): This method is used to initiate the Toast. This method take three parameters:

(a) First is for the application Context,(b) Second is text message and (c)last one is duration for the Toast.

Constants of Toast: Below is the constants of Toast that are used for setting the duration for the Toast.

1. LENGTH_LONG: It is used to display the Toast for a long period of time. When we set this duration the Toast will be displayed for a long duration.

2. LENGTH_SHORT: It is used to display the Toast for short period of time. When we set this duration the Toast will be displayed for short duration.

Below we show the use of makeText() method of Toast in which we set application context, a text message and duration for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast",
Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
```

2. show(): This method is used to display the Toast on the screen. This method is display the text which we create using makeText() method of Toast.

Below we Firstly initiate the Toast and then display it using show() method.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",
Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
```

```
toast.show(); // display the Toast
```

3. setGravity(int,int,int): This method is used to set the gravity for the Toast. This method accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

Below we Firstly initiate the Toast, set top and left gravity and then display it using show() method.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",
Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
```

```
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set
```

```
gravity for the Toast. toast.show(); // display the Toast
```

4. setText(CharSequence s): This method is used to set the text for the Toast. If we use makeText() method and then we want to change the text value for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the text for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",  
Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
```

```
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set gravity for the Toast.
```

```
toast.setText("Changed Toast Text"); // set the text for the Toast
```

```
toast.show(); // display the Toast
```

5. setDuration(int duration): This method is used to set the duration for the Toast. If we use makeText() method and then we want to change the duration for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the duration for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",  
Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
```

```
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set gravity for the Toast.
```

```
toast.setDuration(Toast.LENGTH_SHORT); // set the duration for the Toast.
```

```
toast.show(); // display the Toast
```

6. inflate(int, ViewGroup): This method is used to inflate the layout from the xml. In this method first parameter is the layout resource ID and the second is the root View.

Below we retrieve the Layout Inflater and then inflate the layout from the xml file.

```
// Retrieve the Layout Inflater and inflate the layout from xml
```

```
LayoutInflater inflater = getLayoutInflater();
```

```
View layout = inflater.inflate(R.layout.custom_toast_layout, (ViewGroup)
```

```
        findViewById(R.id.toast_layout_root));
```

7. setView(View): This method is used to set the view for the Toast. In this method we pass the inflated layout which we inflate using inflate() method.

Below we firstly retrieve the layout inflator and then inflate the layout and finally create a new Toast and pass the inflated layout in the setView() method.

```
// Retrieve the Layout Inflater and inflate the layout from xml

LayoutInflater inflater = LayoutInflater.from(this);

View layout = inflater.inflate(R.layout.custom_toast_layout, (ViewGroup)
    findViewById(R.id.toast_layout_root));

// create a new Toast using context

Toast toast = new Toast(getApplicationContext());

toast.setDuration(Toast.LENGTH_LONG); // set the duration for the Toast

toast.setView(layout); // set the inflated layout

toast.show(); // display the custom Toast
```

Custom Toast in Android:

In Android, Sometimes simple Toast may not be satisfactory, and then we can go for customizing a Toast. For creating a custom layout, define a View layout, in XML and pass the root View object to the setView(View) method.

Steps for Implementation of Custom Toast In Android:

Step 1: Firstly Retrieve the Layout Inflater with getLayoutInflator() (or getSystemService()) and then inflate the layout from XML using inflate(int, ViewGroup). In inflate method first parameter is the layout resource ID and the second is the root View.

Step 2: Create a new Toast with Toast(Context) and set some properties of the Toast, such as the duration and gravity.

Step 3: Call setView(View) and pass the inflated layout in this method.

Step 4: Display the Toast on the screen using show() method of Toast.

In the below example we have shown the functioning of Toast and custom Toast both.

Toast And Custom Toast Example In Android Studio:

Below is the example of Toast and Custom Toast in Android. In this example we display two Button's one for Simple Toast and other for Custom Toast and perform click event on them. Whenever a user click on simple Toast Button a Toast with message "Simple Toast In Android"

displayed on the screen and when a user clicks on custom toast Button a message “Custom Toast In Android” with a image displayed on the screen. For Creating a custom toast we firstly retrieve the layout inflator and then inflate the custom toast layout from the xml file. After that we get the reference of TextView and ImageView from the inflated layout and set the text and image in the TextView and ImageView. Finally we create a new Toast and pass the inflated layout in the setView() method and then display the Toast by using show() method of Toast.



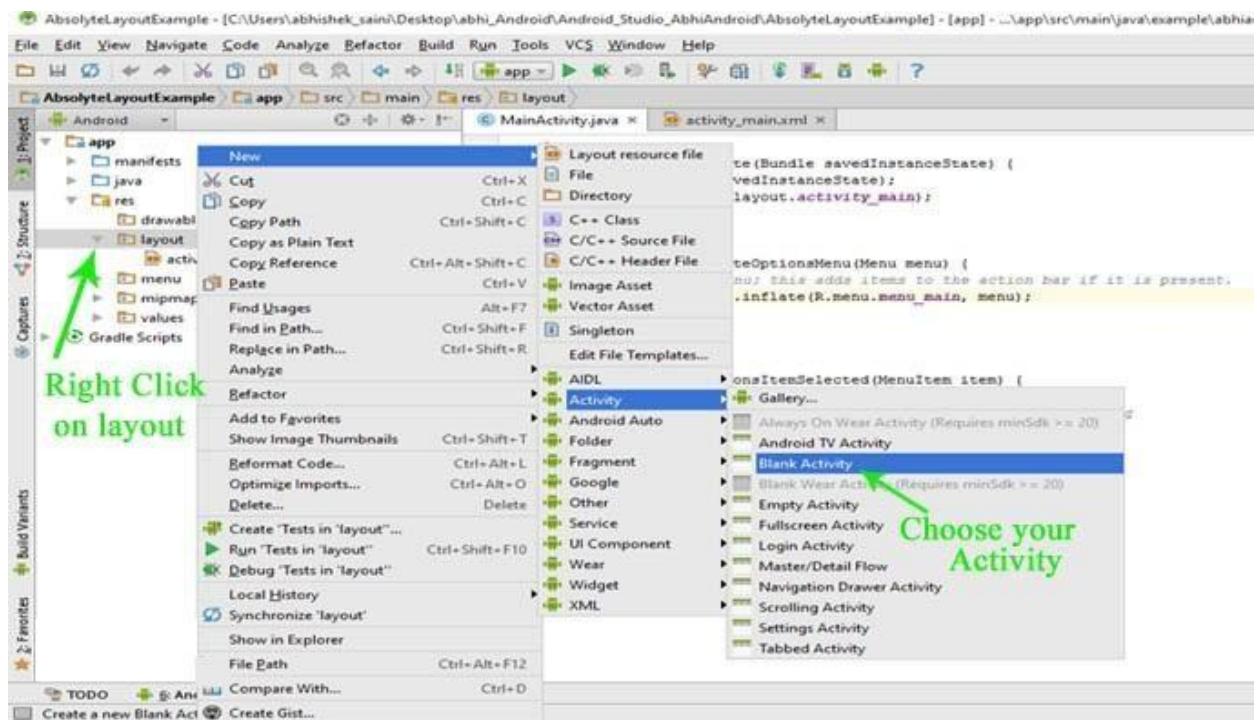
3.5 CREATING AND STARTING AN ACTIVITY

How To Create New Activity in Android Studio

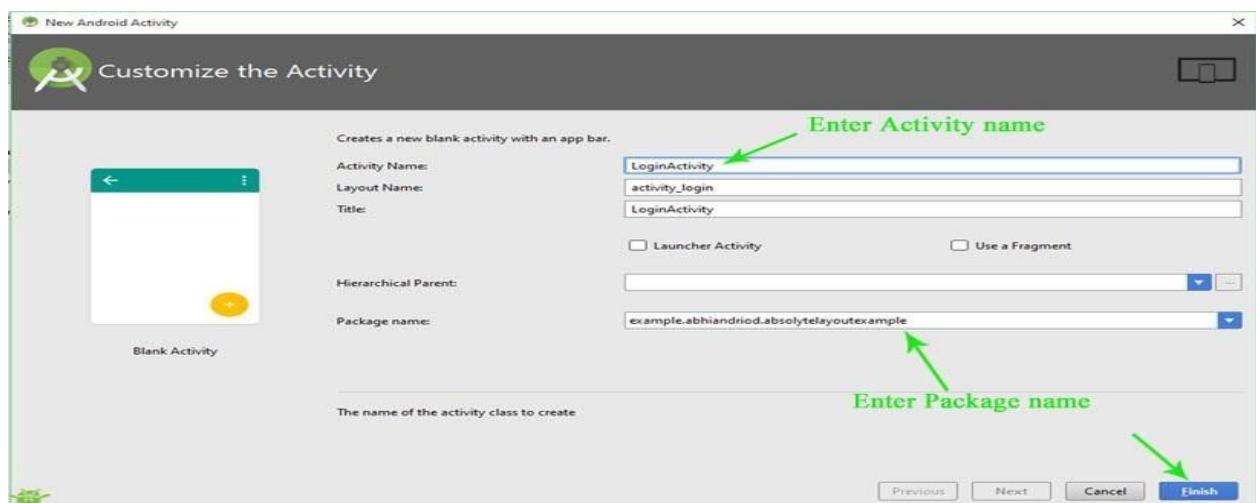
We create New Activity in Android Studio to create XML file for designing UI and java file coding. Below are the steps to create new Activity in Android Studio:

How To Create New Activity in Android Studio:

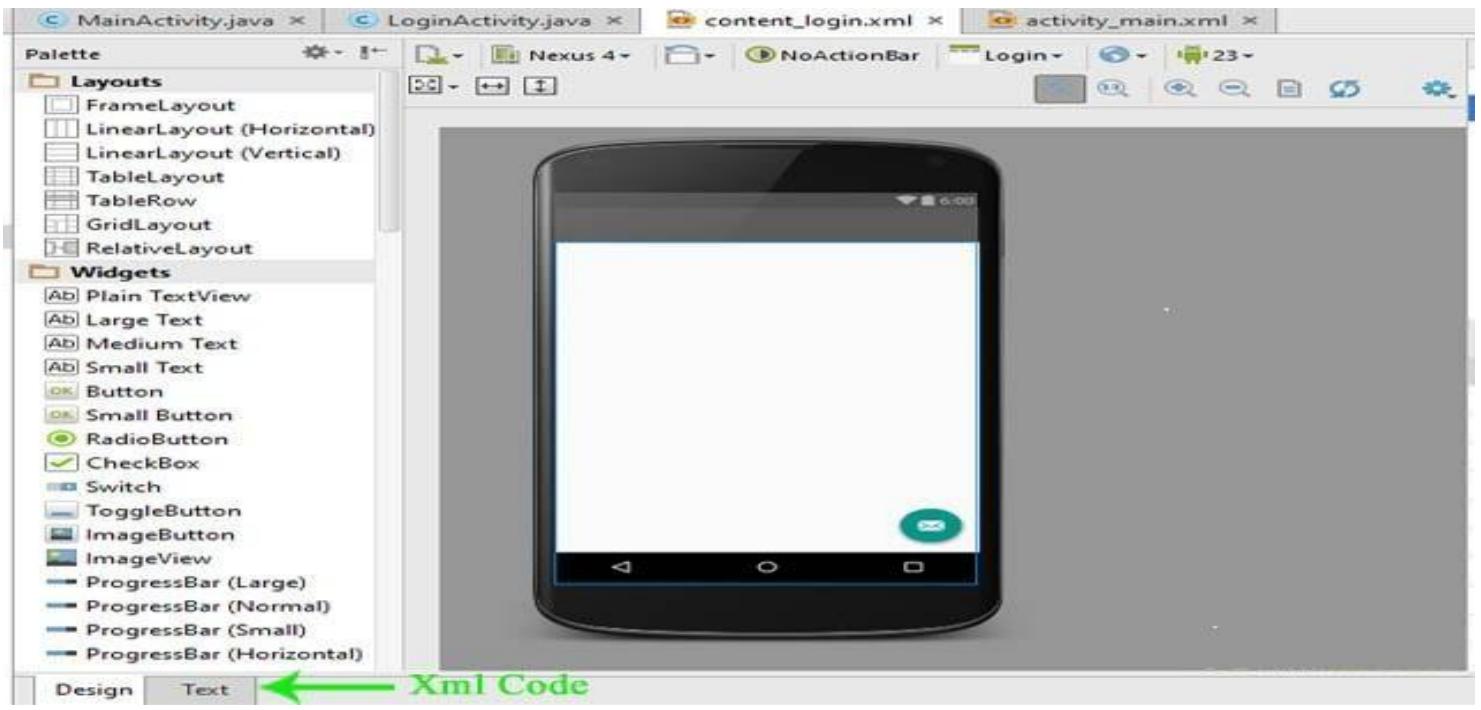
Step 1: Firstly, click on app > res > layout > Right Click on layout. After that Select New > Activity and choose your Activity as per requirement. Here we choose Blank Activity as shown in figure below.



Step 2: After that Customize the Activity in Android Studio. Enter the “Activity Name” and “Package name” in the Text box and Click on Finish button.



Step 3: After that your new Activity in Layout will be created. Your XML Code is in Text and your Design Output is in Design.



3.6 USING THE EDIT TEXT CONTROL

EditText Tutorial With Example In Android Studio: Input Field

In Android, EditText is a standard entry widget in android apps. It is an overlay over TextView that configures itself to be editable. EditText is a subclass of TextView with text editing operations. **We often use EditText in our applications in order to provide an input or text field, especially in forms.** The most simple example of EditText is Login or Sign-in form.



Text Fields in Android Studio are basically EditText:



Important Note: An EditText is simply a thin extension of a TextView. An EditText inherits all the properties of a TextView.

EditText Code:

We can create a EditText instance by declaring it inside a layout(XML file) or by instantiating it programmatically (i.e. in Java Class).

EditText code in XML:

```
<EditText android:id="@+id/simpleEditText" android:layout_height="wrap_content"
        android:layout_width="match_parent"/>
```

Retrieving / Getting the Value From EditText In Java Class:

Below is the example code of EditText in which we retrieve the value from a EditText in Java class. We have used this code in the example you will find at the end of this post.

```
EditText simpleEditText = (EditText) findViewById(R.id.simpleEditText);
```

```
String editTextValue = simpleEditText.getText().toString();
```

Attributes of EditText:

Now let's we discuss few attributes that helps us to configure an EditText in your xml.

1. id: id is an attribute used to uniquely identify a text EditText. Below is the example code in which we set the id of a edit text.

```
<EditText android:id="@+id/simpleEditText"
```

```
        android:layout_height="wrap_content"
```

```
        android:layout_width="match_parent"/>
```

2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below is the example code with explanation included in which we set the right gravity for text of a EditText.

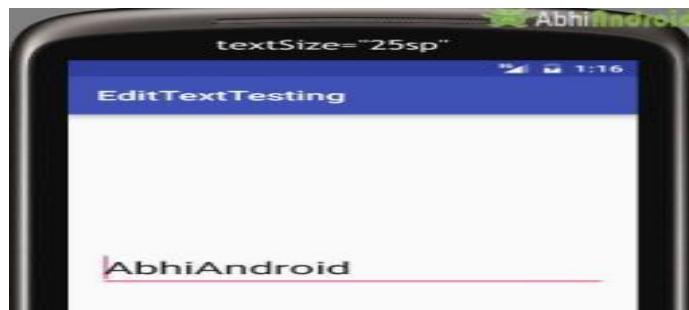
```
<EditText android:id="@+id/simpleEditText" android:layout_width="fill_parent"  
        android:layout_height="wrap_content" android:text="Enter Email"  
        android:gravity="right"/><!--gravity of a edit text-->
```



3. text: text attribute is used to set the text in a EditText. We can set the text in xml as well as in the java class.

Below is the example code in which we set the text “Username” in a edit text.

```
<EditText  
        android:id="@+id/simpleEditText"  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:layout_centerInParent="true"  
        android:text="Username"/><!--set text in edit text-->
```



Setting text in EditText In Java class:

Below is the example code in which we set the text in a text view programmatically means in java class.

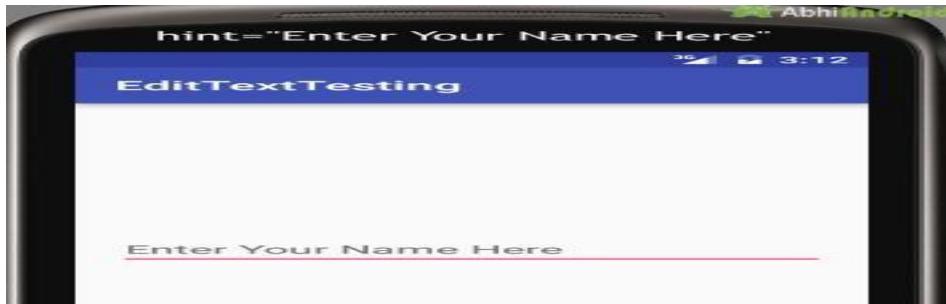
```
EditText editText = (EditText)findViewById(R.id.simpleEditText);
```

```
editText.setText("Username");//set the text in edit text
```

4. hint: hint is an attribute used to set the hint i.e. what you want user to enter in this edit text. Whenever user start to type in edit text the hint will automatically disappear.

Below is the example code with explanation in which we set the hint of a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:hint="Enter Your Name Here" /><!--display the hint-->
```



Setting hint in EditText In Java class:

Below is the example code in which we set the text in a text view programmatically means in java class.

```
EditText editText = (EditText)findViewById(R.id.simpleEditText);  
editText.setHint("Enter Your Name Here");//display the hint
```

5. textColor: textColor attribute is used to set the text color of a text edit text. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below is the example code with explanation included in which we set the red color for the displayed text of a edit text.

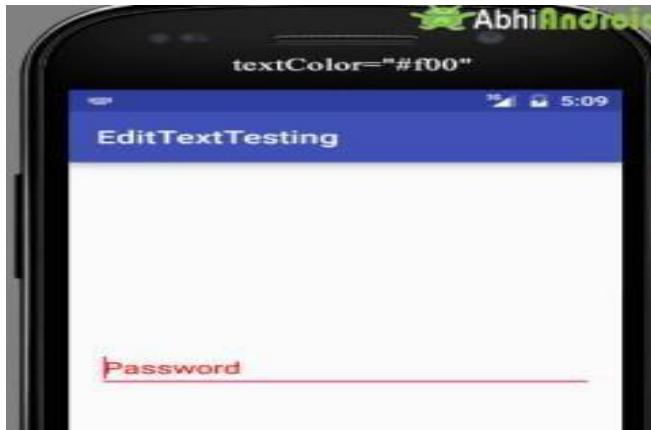
```
<EditText android:id="@+id/simpleEditText"
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="Password"
    android:textColor="#f00"/><!--set the red text color--&gt;</pre>
```

Setting textColor in EditText In Java class:

Below is the example code in which we set the text color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setTextColor(Color.RED);//set the red text color
```



6. textColorHint: textColorHint is an attribute used to set the color of displayed hint.

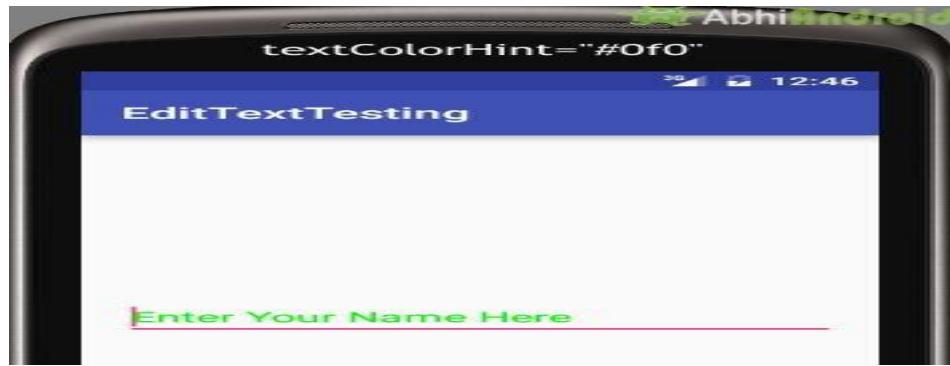
Below is the example code with explanation included in which we set the green color for displayed hint of a edit text.

```
<EditText
    android:id="@+id/simpleEditText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:hint="Enter Your Name Here"
    android:textColorHint="#0f0"/><!--set the hint color green--&gt;</pre>
```

Setting textColorHint in EditText In Java class:

Below is the example code in which we set the hint color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setHintTextColor(Color.green(0));//set the green hint color
```



7. textSize: textSize attribute is used to set the size of text of a edit text. We can set the text size in sp(scale independent pixel) or dp(density pixel).

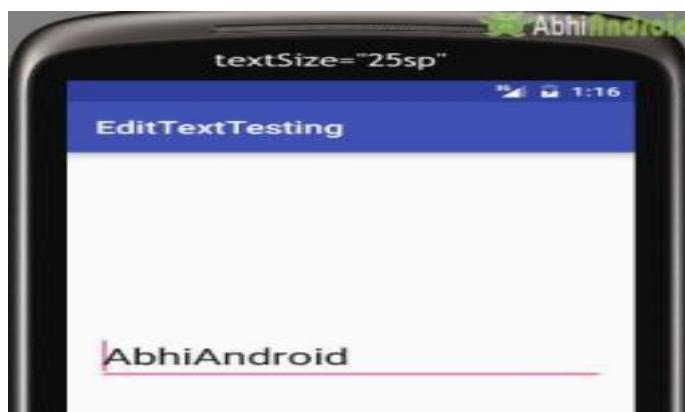
Below is the example code in which we set the 25sp size for the text of a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:text="AbhiAndroid"  
    android:textSize="25sp" /><!--set 25sp text size-->
```

Setting textSize in EditText in Java class:

Below is the example code in which we set the text size of a edit text programmatically means in java class.

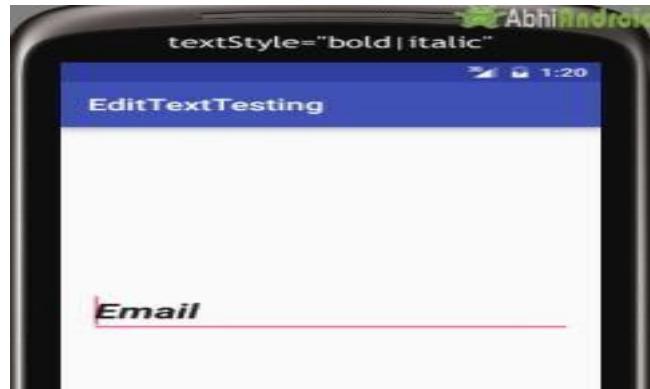
```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);  
simpleEditText.setTextSize(25); //set size of text
```



8. textStyle: textStyle attribute is used to set the text style of a edit text. The possible text styles are bold, italic and normal. If we need to use two or more styles for a edit text then “|” operator is used for that.

Below is the example code with explanation included, in which we set the bold and italic text styles for text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:text="Email" android:textSize="25sp"  
    android:textStyle="bold|italic"/><!--set bold and italic text style-->
```



9. background: background attribute is used to set the background of a edit text. We can set a color or a drawable in the background of a edit text.

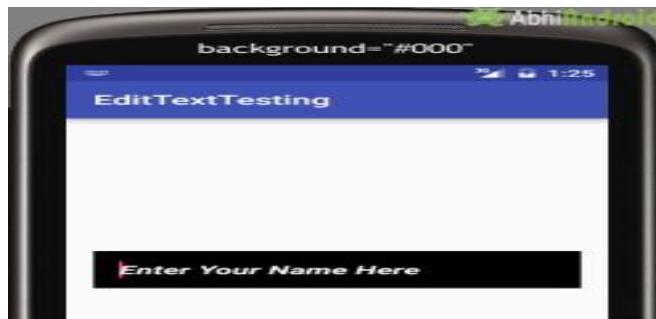
Below is the example code with explanation included in which we set the black color for the background, white color for the displayed hint and set 10dp padding from all the side's for edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:hint="Enter Your Name Here"  
    android:padding="15dp" android:textColorHint="#fff"  
    android:textStyle="bold|italic"  
    android:background="#000"/><!--set background color black-->
```

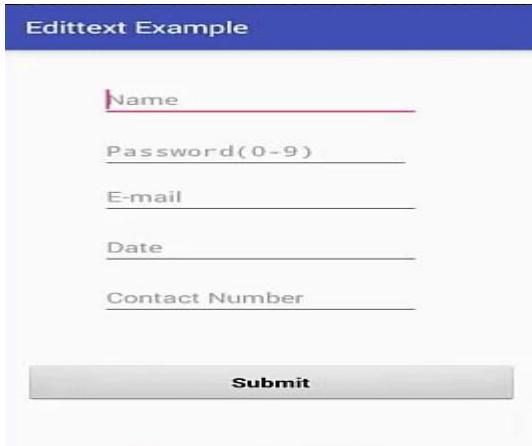
Setting Background in EditText In Java class: Below is the example code in which we set the background color of a edit text programmatically means in java class.

EditText

```
simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setBackgroundColor(Color.BLACK);//set black
background color
```



10. padding: padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of edit text.



Example I – EditText in Android Studio

Below is the example of edit text in which we get the value from multiple edittexts and on button click event the Toast will show the data defined in Edittext.

Step 1: Create a new project in Android Studio and name it EditTextExample. **Step 2:** Now Open res -> layout -> xml (or) activity_main.xml and add following code. In this code we have added multiple edittext and a button with onclick functionality.

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.edittextexample.MainActivity">
<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"

    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="50dp"
    android:layout_marginStart="50dp"
    android:layout_marginTop="24dp"
    android:ems="10"
    android:hint="@string/name"
    android:inputType="textPersonName"
    android:selectAllOnFocus="true" />
<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_alignStart="@+id/editText1"
    android:layout_below="@+id/editText1"
    android:layout_marginTop="19dp"
    android:ems="10"
    android:hint="@string/password_0_9"
    android:inputType="numberPassword" />
<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText2"
    android:layout_alignStart="@+id/editText2"
    android:layout_below="@+id/editText2"
    android:layout_marginTop="12dp"
    android:ems="10"
```

```

        android:hint="@string/e_mail"
        android:inputType="textEmailAddress" />
<EditText
    android:id="@+id/editText4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText3"
    android:layout_alignStart="@+id/editText3"
    android:layout_below="@+id/editText3"
    android:layout_marginTop="18dp"
    android:ems="10" android:hint="@string/date"
    android:inputType="date" />
<EditText
    android:id="@+id/editText5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText4"
    android:layout_alignStart="@+id/editText4"
    android:layout_below="@+id/editText4"
    android:layout_marginTop="18dp"
    android:ems="10"
    android:hint="@string/contact_number"
    android:inputType="phone" />
<Button
    android:id="@+id/button"
    style="@android:style/Widget.Button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_below="@+id/editText5"
    android:layout_marginTop="62dp"
    android:text="@string/submit"
    android:textSize="16sp"
    android:textStyle="normal|bold" />
</RelativeLayout>
```

Step 3: Now open app -> java -> package -> MainActivity.java and add the below code. In this we just fetch the text from the edittext, further with the button click event a toast will show the text fetched before.

```
package com.example.edittextexample;
```

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends
    AppCompatActivity { Button submit;
    EditText name, password, email, contact, date;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = (EditText) findViewById(R.id.editText1);
        password = (EditText)
            findViewById(R.id.editText2);
        email = (EditText) findViewById(R.id.editText3);
        date = (EditText) findViewById(R.id.editText4);
        contact = (EditText)
            findViewById(R.id.editText5);
        submit = (Button) findViewById(R.id.button);
        submit.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v) {
                if (name.getText().toString().isEmpty() || password.getText().toString().isEmpty() ||
                    email.getText().toString().isEmpty() || date.getText().toString().isEmpty()
                    || contact.getText().toString().isEmpty()) {
                    Toast.makeText(getApplicationContext(), "Enter the Data",
                        Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getApplicationContext(), "Name - " + name.getText().toString() + "\n"
                        + "Password - " + password.getText().toString() + "\n" + "E-Mail - " +
                        email.getText().toString() + "\n" + "Date - " + date.getText().toString() + "\n" +
                        "Contact - " + contact.getText().toString(),
                        Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}

```

```
});
```

```
}
```

```
}
```

Output:

Now start the AVD in Emulator and run the App. You will see screen asking you to fill the data in required fields like name, password(numeric), email, date, contact number. Enter data and click on button. You will see the data entered will be displayed as Toast on screen.

Below is the example of edit text in which we get the value from a edit text on button click event and then display it in a Toast. Below is the final output and code.



Example II – EditText in Android Studio



Step 1: Create a new project in Android Studio and name it **EditTextExample**.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Now Open res -> layout -> xml (or) activity_main.xml and add following code. Here we will design one EditText for filling name and one Button which will be used to display the name entered by the user.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <EditText android:id="@+id/simpleEditText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:background="#F2F2F2"
        android:hint="Enter Your Name Here"
        android:padding="15dp"
        android:textColorHint="#000"
        android:textStyle="bold|italic"
        android:layout_marginTop="100dp" />
```

```

<Button
    android:id="@+id/displayText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:background="#000"
    android:padding="10dp"
    android:text="Display Text"
    android:textColor="#0f0"
    android:textStyle="bold" />
</RelativeLayout>

```

Step 3: Now open app -> java -> package -> MainActivity.java and add the below code. The explanation is included in the code itself as comment.

```

package
example.abhiandriod.edittextexample;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends
    AppCompatActivity { @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText simpleEditText = (EditText)
            findViewById(R.id.simpleEditText);//get the id for edit text Button
        displayText = (Button) findViewById(R.id.displayText);//get the id for button
        displayText.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (simpleEditText.getText().toString() != null)//check whether the entered text is not null
                {
                    Toast.makeText(getApplicationContext(), simpleEditText.getText().toString(),

```

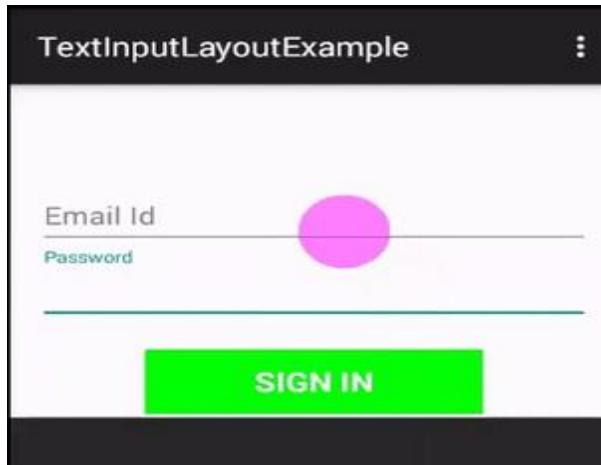
```
        Toast.LENGTH_LONG).show();//display the text that you entered in edit text  
    }  
  
}  
  
});  
  
}  
  
}  
  
}
```

Output: Now start the AVD in Emulator and run the App. You will see screen asking you to fill your name. Enter your name and click on button. You will see the name entered will be displayed as Toast on screen.



TextInputLayout / Floating Labels in EditText:

TextInputLayout is a new element introduced in Material Design Support library to display the floating label in EditText. Read our advance Floating Labels tutorial to learn how to use it in your App.



3.7 CHOOSING OPTIONS WITH CHECKBOX

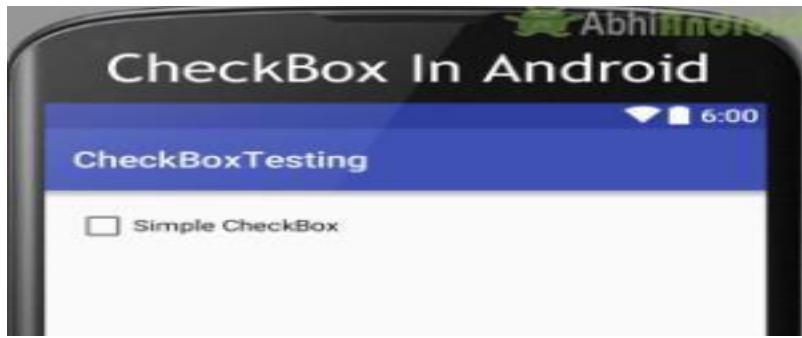
In Android, CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBox class.



In android there is a lot of usage of check box. For example, to take survey in Android app we can list few options and allow user to choose using CheckBox. The user will simply checked these checkboxes rather than type their own option in EditText. Another very common use of CheckBox is as remember me option in Login form.

CheckBox code in XML:

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Simple CheckBox"/>
```



Important Note: You can check the current state of a check box programmatically by using `isChecked()` method. This method returns a Boolean value either true or false, if a check box is checked then it returns true otherwise it returns false. Below is an example code in which we checked the current state of a check box.

```
//initiate a check box  
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);  
//check current state of a check box (true or false) Boolean  
checkBoxState = simpleCheckBox.isChecked();
```

Attributes of CheckBox:

Now let's we discuss important attributes that helps us to configure a check box in XML file (layout).

1. id: id is an attribute used to uniquely identify a check box. Below we set the id of a image button.

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Simple CheckBox"/>
```

2. checked: checked is an attribute of check box used to set the current state of a check box. The value should be true or false where true shows the checked state and false shows unchecked state of a check box. The default value of checked attribute is false. We can also set the current state programmatically.

Below is an example code in which we set true value for checked attribute that sets the current state to checked.

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Simple CheckBox"  
    android:checked="true"/> <!--set the current state of the check box-->
```



Setting Current State Of CheckBox In Java Class:

Below we set the current state of CheckBox in java class.

```
/*Add in Oncreate() funtion after setContentView()*/
```

```
// initiate a check box
```

```
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
```

```
// set the current state of a check box
```

```
simpleCheckBox.setChecked(true);
```

3. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text in CheckBox like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below we set the right and center_vertical gravity for the text of a check box.

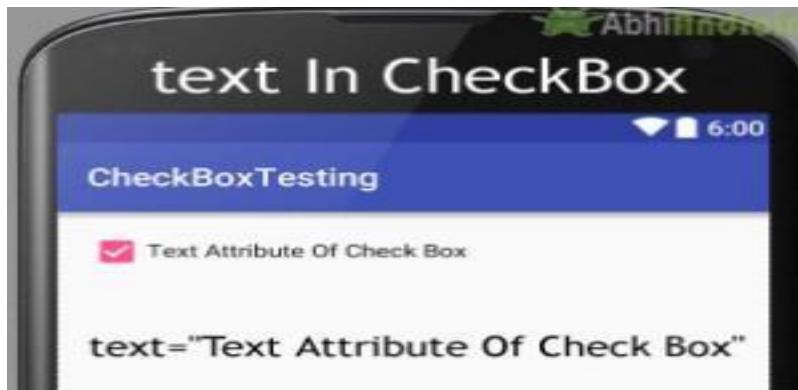
```
<CheckBox android:id="@+id/simpleCheckBox"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Simple CheckBox"
    android:checked="true"
    android:gravity="right|center_vertical"/><!-- gravity of the check box-->
```



4. text: text attribute is used to set the text in a check box. We can set the text in xml as well as in the java class.

Below is the example code with explanation included in which we set the text “Text Attribute Of Check Box” for a check box.

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:text="Text Attribute Of Check Box"/> <!--displayed text of the check box-->
```



Setting text in CheckBox In Java class:

Below is the example code in which we set the text of a check box programmatically means in java class.

```
/*Add in Oncreate() function after setContentView()*/
```

```
// initiate check box
```

```
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
```

```
// displayed text of the check box
```

```
simpleCheckBox.setText("Text Attribute Of Check Box");
```

5. textColor: textColor attribute is used to set the text color of a check box. Color value is in form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below we set the red color for the displayed text of a check box.

```
<CheckBox
```

```
    android:id="@+id/simpleCheckBox"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Text is Red Color"
android:textColor="#f00"
android:checked="true"/> <!-- red color for the text of check box-->
```



Setting textColor in CheckBox In Java class:

Below we set the text color of a check box programmatically.

```
/*Add in Oncreate() function after setContentView()*/
```

```
//initiate the checkbox
```

```
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
```

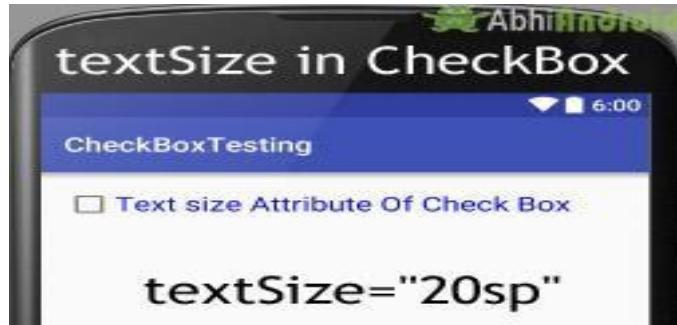
```
//red color for displayed text
```

```
simpleCheckBox.setTextColor(Color.RED);
```

6. textSize: textSize attribute is used to set the size of text of a check box. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 20sp size for the text of a check box.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text size Attribute Of Check Box"
    android:textColor="#00f"
    android:checked="false"
    android:textSize="20sp"/><!--set Text Size of text in CheckBox-->
```



Setting Text Size in CheckBox In Java class:

Below we set the text size of a check box in java class.

```
/*Add in Oncreate() funtion after setContentView()*/
```

```
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
```

```
//set 20sp displayed text size
```

```
simpleCheckBox.setTextSize(20);
```

7. textStyle: textStyle attribute is used to set the text style of the text of a check box. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then “|” operator is used for that.

Below we set the bold and italic text styles for text of a check box.

```
<Check Box android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text Style Attribute Of Check Box"
    android:textColor="#44f"
    android:textSize="20sp"
    android:checked="false"
    android:textStyle="bold|italic"/>
```

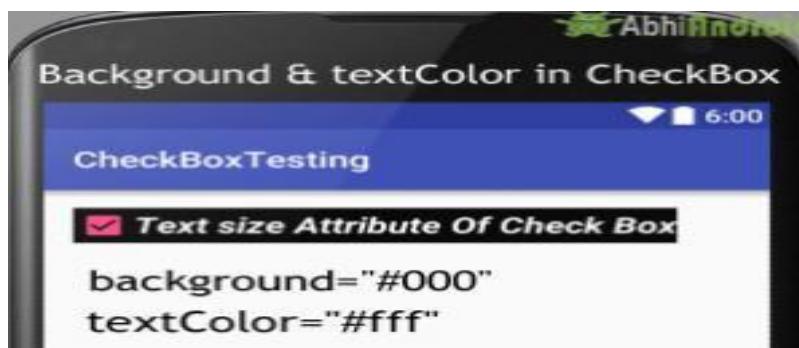


8. background: background attribute is used to set the background of a check box. We can set a color or a drawable in the background of a check box.

Below we set the black color for the background, white color for the displayed text of a check box.

<Check Box

```
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text size Attribute Of Check Box"
    android:textColor="#fff" android:textSize="20sp"
    android:textStyle="bold|italic"
    android:checked="true"
    android:background="#000" /><!-- black background for the background of check box-->
```



Setting Background in CheckBox In Java class:

Below is the example code in which we set the background color of a check box programmatically means in java class.

```
/*Add in Oncreate() function after setContentView()*/
```

```
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
```

```
// set background in CheckBox
```

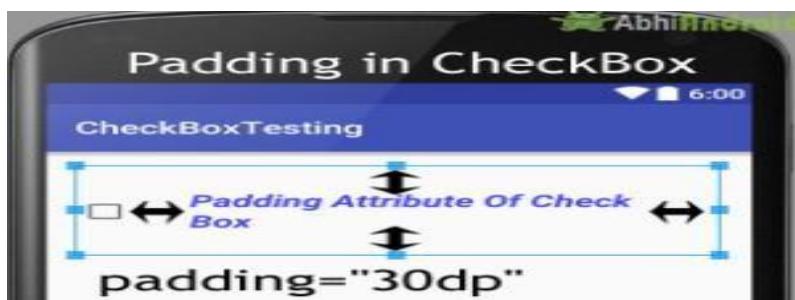
```
simpleCheckBox.setBackgroundColor(Color.BLACK);
```

9. padding: padding attribute is used to set the padding from left, right, top or bottom.

- **paddingRight** :set the padding from the right side of the check box.
- **paddingLeft** :set the padding from the left side of the check box.
- **paddingTop** :set the padding from the top side of the check box.
- **paddingBottom** :set the padding from the bottom side of the check box.
- **Padding** :set the padding from the all side's of the check box.

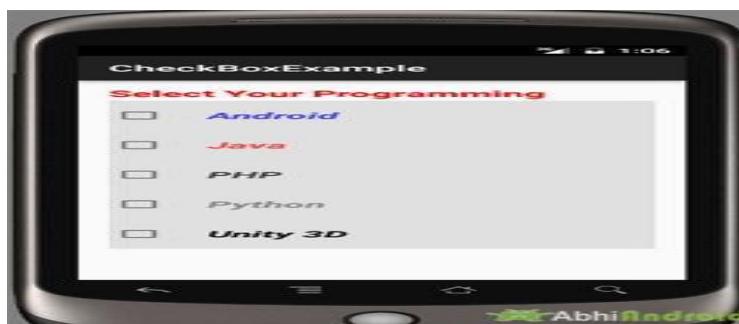
Below is the example code of padding where we set the 30dp padding from all the side's of the check box.

```
<Check Box  android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Padding Attribute Of Check Box"
    android:textColor="#44f"
    android:textSize="20sp"
    android:textStyle="bold|italic"
    android:checked="false"
    android:padding="30dp"/> <!--30dp padding from all side's-->
```



Example of CheckBox In Android Studio:

Below is the example of CheckBox in Android, in which we display five check boxes using background and other attributes we discusses earlier in this post. Every check box represents a different subject name and whenever you click on a check box then text of that checked check box is displayed in a toast. Below is the final output, code and step by step explanation of the example:



Step 1: Create a new project and name it CheckBoxExample

In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Now Open res -> layout -> **activity_main.xml (or) main.xml** and add the following code:

In this step we open an xml file and add the code for displaying five one TextView and check boxes.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <Text View
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Your Programming language: "
        android:textColor="#f00"
        android:textSize="20sp"
        android:textStyle="bold" />
    <Linear Layout
        android:id="@+id/linearLayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:background="#e0e0e0"
        android:orientation="vertical">
        <Check Box
            android:id="@+id/androidCheckBox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:checked="false"
```

```
        android:padding="20dp"
        android:text="@string/android"
        android:textColor="#44f"
        android:textSize="20sp"
        android:textStyle="bold|italic" />
<Check Box
    android:id="@+id/javaCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:checked="false"
    android:padding="20dp"
    android:text="@string/java"
    android:textColor="#f44"
    android:textSize="20sp"
    android:textStyle="bold|italic" />
<Check Box
    android:id="@+id/phpCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:checked="false"
    android:padding="20dp"
    android:text="@string/php"
    android:textColor="#444"
    android:textSize="20sp"
    android:textStyle="bold|italic" />
<Check Box
    android:id="@+id/pythonCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:checked="false"
    android:padding="20dp"
    android:text="@string/python"
    android:textColor="#888"
    android:textSize="20sp"
    android:textStyle="bold|italic" />
<Check Box
    android:id="@+id/unityCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
```

```

        android:checked="false"
        android:padding="20dp"
        android:text="@string/unity"
        android:textColor="#101010"
        android:textSize="20sp"
        android:textStyle="bold|italic" />
    </LinearLayout>

</RelativeLayout>
```

Step 3: Now Open app -> java-> package -> **MainActivity.java**

In this step we add the code to initiate the check boxes we created. And then we perform click event on button and display the text for selected check boxes using a toast.

package example.

```

andriod.checkboxexample;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity implements View.OnClickListener
{
    CheckBox android, java, python, php, unity3D;
    @Override

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate views

        android = (CheckBox)
        findViewById(R.id.androidCheckBox);
        android.setOnClickListener(this);
        java = (CheckBox) findViewById(R.id.javaCheckBox);

        java.setOnClickListener(this);
```

```
python = (CheckBox)
findViewById(R.id.pythonCheckBox);
python.setOnClickListener(this);
php = (CheckBox)
findViewById(R.id.phpCheckBox);
php.setOnClickListener(this);
unity3D = (CheckBox)
findViewById(R.id.unityCheckBox);
unity3D.setOnClickListener(this);
}

@Override

public void
onClick(View view) {
switch (view.getId()) {
case R.id.androidCheckBox:
if (android.isChecked())
Toast.makeText(getApplicationContext(), "Android", Toast.LENGTH_LONG).show();
break;
case R.id.javaCheckBox:
if (java.isChecked())
Toast.makeText(getApplicationContext(), "Java", Toast.LENGTH_LONG).show();
break;
case R.id.phpCheckBox:
if (php.isChecked())
Toast.makeText(getApplicationContext(), "PHP", Toast.LENGTH_LONG).show();
break;
case R.id.pythonCheckBox:
if (python.isChecked())
Toast.makeText(getApplicationContext(), "Python", Toast.LENGTH_LONG).show();
break;
case R.id.unityCheckBox:
if (unity3D.isChecked())
Toast.makeText(getApplicationContext(), "Unity 3D", Toast.LENGTH_LONG).show();
break;
}
}

}
```

Step 5: Open res ->values -> strings.xml

In this step we show string file which is used to store string data of an app.

```
<resources>
    <string name="app_name">CheckBoxExample</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="android">Android</string>
    <string name="java">Java</string>
    <string name="php">PHP</string>
    <string name="python" >Python</string>
    <string name="unity">Unity 3D</string>
</resources>
```

Output: Now start the AVD in Emulator and run the App. You will see 5 checkbox option asking you to choose your programming language. Select and the text of that particular CheckBox will appear on Screen.



3.8 CHOOSING MUTUALLY EXCLUSIVE ITEMS USING RADIO BUTTONS

In Android, RadioButton are mainly used together in a RadioGroup. In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group. The most common use of radio button is in Quiz App.



RadioButton is a two state button that can be checked or unchecked. If a radio button is unchecked then a user can check it by simply clicking on it. Once a RadioButton is checked by user it can't be unchecked by simply pressing on the same button. It will automatically unchecked when you press any other RadioButton within same RadioGroup.

Important Note: RadioGroup is a widget used in Android for the grouping of radio buttons and provide the feature of selecting only one radio button from the set. When a user tries to select any other radio button within same radio group the previously selected radio button will be automatically unchecked.

Radio Group And Radio Button code in XML:

```
<Radio Group  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content">  
    <Radio Button  
        android:id="@+id/simpleRadioButton"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"/>  
    <Radio Button  
        android:id="@+id/simpleRadioButton1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"/>  
</Radio Group>
```



Checking Current State of Radio Button:

You can check the current state of a radio button programmatically by using isChecked() method. This method returns a Boolean value either true or false. if it is checked then returns true otherwise returns false. Below is an example code with explanation in which we checked the current state of a radio button.

```
/*Add in Oncreate() function after setContentView()*/
```

```
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);  
// initiate a radio button Boolean RadioButtonState = simpleRadioButton.isChecked(); //  
check current state of a radio button (true or false).
```

Attributes of RadioButton In Android:

Now let's we discuss important attributes that helps us to create a beautiful radio button in xml file (layout).

1.id: id is an attribute used to uniquely identify a radio button.

```
<Radio Button  
    android:id="@+id/simpleRadioButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```

2.**checked**: checked attribute in radio button is used to set the current state of a radio button.

We can set it either true or false where true shows the checked state and false shows unchecked state of a radio button. As usual default value of checked attribute is false. We can also set the current state in JAVA.

Below we set true value for checked attribute which sets the current state to checked of a Button

```
<Radio Button  
    android:id="@+id/simpleRadioButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"/> <!-- set the current state of the radio button-->
```



Setting checked State Of RadioButton In Java Class:

Below code set the current state of RadioButton to checked programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/  
  
// initiate a radio button  
  
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);  
  
// set the current state of a radio  
button  
  
simpleRadioButton.setChecked  
(true);
```

3.text: text attribute is used to set the text in a radio button. We can set the text both ways either in XML or in JAVA class.

Below is the example code with explanation included in which we set the text “I am a radiobutton” of a radio button.

```
<RadioButton  
    android:id="@+id/simpleRadioButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:layout_centerHorizontal="true"  
    android:text="I am a radiobutton" /><!-- displayed text of radio button-->
```

Setting text of RadioButton In Java class:

Below we set the text of a radio button programmatically:

```
/*Add in Oncreate() funtion after setContentView()*/  
  
RadioButton simpleRadioButton=(RadioButton)  
findViewById(R.id.simpleRadioButton);  
simpleRadioButton.setText("I am a radiobutton"); // displayed text of  
radio button
```



4.gravity: The gravity attribute is an optional attribute which is used to control the alignment of text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below is the example code with explanation included in which we set the center gravity for the text of a radio button.

```
<Radio Button android:id="@+id/simpleRadioButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:checked="true"
    android:text="I am a Button"
    android:gravity="center"/><!-- center gravity of the text-->
```



5.textColor: textColor attribute is used to set the text color of a radio button. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

Below we set the red color for the displayed text of a radio button.

```
<Radio Button
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:layout_centerHorizontal="true"
    android:text="Male"
    android:textColor="#f00" /><!--red color for displayed text-->
```

Setting text Color of Radio Button text In Java class:

Below we set the text color of a radio button programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/
```

```
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);
// initiate radio button
simpleRadioButton.setTextColor(Color.RED); //red color for displayed text of radio button
```



6.text Size: text Size attribute is used to set the size of the text of a radio button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below we set the 25sp size for the text of a radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:layout_centerHorizontal="true"
    android:text="AbhiAndroid"
    android:textColor="#f00"
    android:textSize="25sp"/> <!--setting
text size-->
```



Setting textSize Of RadioButton Text In Java class:

Below we set the text size of a radio button programmatically:

```
/*Add in Oncreate() funtion after setContentView()*/
```

```
RadioButton simpleRadioButton = (RadioButton)
```

```
findViewById(R.id.simpleRadioButton); // initiate radio button
```

```
simpleRadioButton.setTextSize(25); // set 25sp displayed text size of radio button
```

7.textStyle: textStyle attribute is used to set the text style of the text of a radio button. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then “|” operator is used for that.

Below is the example code with explanation included in which we set the bold and italic text styles for text of a radio button.

```
<Radio Button  
    android:id="@+id/simpleRadioButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:textSize="25sp"  
    android:layout_centerHorizontal="true"  
    android:text="Male"  
    android:textColor="#f00"  
    android:textStyle="bold|italic"/> <!-- bold and italic text style-->
```



8.background: background attribute is used to set the background of a radio button. We can set a color or a drawable in the background of a radio button.

Below we set the black color for the background and red color for the displayed text of a radio button.

```
<RadioButton
```

```
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textSize="25sp"
    android:textStyle="bold|italic"
    android:padding="20dp"
    android:layout_centerHorizontal="true"
    android:text="Male"
    android:textColor="#f00"
    android:background="#000"/> <!-- black background for radio button-->
```

Setting Background of RadioButton in Java class:

Below we set the background color of a radio button programmatically.

```
/*Add in Oncreate() function after setContentView()*/
```

```
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);
simpleRadioButton.setBackgroundColor(Color.BLACK);
```



9.padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight: set the padding from the right side of the radio button.

paddingLeft : set the padding from the left side of the radio button.

paddingTop : set the padding from the top side of the radio button.

paddingBottom: set the padding from the bottom side of the radio button.

Padding: set the padding from the all sides of the radio button.

Below we set padding attribute of 20dp padding from all the side's of the radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:checked="true" android:textSize="25sp"
    android:textStyle="bold|italic"
    android:layout_centerHorizontal="true"
    android:text="AbhiAndroid"
    android:textColor="#f00"
    android:padding="40dp"/> <!--40dp padding from all the sides of radio button-->
```



10. drawableBottom, drawableTop, drawableLeft And drawableRight: These attribute draw the drawable to the below of the text of RadioButton.

Below we set the icon to the right of the text of a RadioButton.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textSize="25sp"
    android:padding="20dp"
    android:layout_centerHorizontal="true"
    android:text="AbhiAndroid"
    android:textColor="#f00"
    android:drawableRight="@drawable/ic_launcher" /> <!-- drawable icon at the right of radio button-->
```



Example Of RadioButton And RadioGroup in Android Studio:

Below is the example of Radiobutton in Android where we display five radio buttons with background and other attributes. The radio buttons are used to select your favorite WWE superstar with one “submit” button. Below is the final output, download code and step by step explanation of tutorial:



Step 1: Create a new project and name it Radio Button Example

Select File -> New -> New Project and Fill the forms and click “Finish” button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying 5 Radio Button and one normal button.

<LinearLayout

```
xmlns:android="http://schemas.android.com/apk/res/android
" xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:paddingBottom="@dimen/activity_vertical_margin"
```

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
<Linear Layout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:orientation="vertical">

    <Text View
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select your favourite wwe
        SuperStar :: " android:textColor="#000"
        android:textSize="20sp"
        android:textStyle="bold" />
    <Radio Group
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Radio Button
            android:id="@+id/johnCena"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="20dp"
            android:layout_marginTop="10dp"
            android:checked="true"
            android:text="@string/johnCena"
            android:textColor="#154"
            android:textSize="20sp"
            android:textStyle="bold" />
        <Radio Button
            android:id="@+id/randyOrton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="20dp"
            android:layout_marginTop="10dp"
            android:checked="false"
            android:text="@string/randyOrton"
            android:textColor="#154"
            android:textSize="20sp"
            android:textStyle="bold" />
        <Radio Button
```

```
    android:id="@+id/romanReigns"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/romanReigns"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />
<Radio Button
    android:id="@+id/goldBerg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/goldBerg"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />
<Radio Button android:id="@+id/sheamus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/sheamus"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />
</RadioGroup>
<Button android:id="@+id/submitButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="20dp"
    android:background="#0f0"
    android:padding="10dp"
    android:text="Submit"
    android:textColor="#fff"
    android:textSize="20sp"
    android:textStyle="bold" />
```

```
</LinearLayout>
```

```
</LinearLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the RadioButton and normal button. We also perform click event on button and display the selected superstar's name by using a Toast.

```
package example.gb.radiobuttonexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
    RadioButton johnCena, randyOrton, goldBerg, romanReigns, sheamus;
    String selectedSuperStar;
    Button submit;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        johnCena = (RadioButton) findViewById(R.id.johnCena);
        randyOrton = (RadioButton) findViewById(R.id.randyOrton);
        goldBerg = (RadioButton) findViewById(R.id.goldBerg);
        romanReigns = (RadioButton) findViewById(R.id.romanReigns);
        sheamus = (RadioButton) findViewById(R.id.sheamus);
        submit = (Button) findViewById(R.id.submitButton);
        submit.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v) {
                if (randyOrton.isChecked()) {
                    selectedSuperStar = randyOrton.getText().toString();
                } else if (sheamus.isChecked()) {
                    selectedSuperStar = sheamus.getText().toString();
                } else if (johnCena.isChecked()) {
                    selectedSuperStar = johnCena.getText().toString();
                } else if (romanReigns.isChecked()) {
                    selectedSuperStar = romanReigns.getText().toString();
                } else if (goldBerg.isChecked()) {
                    selectedSuperStar = goldBerg.getText().toString();
                }
            }
        });
    }
}
```

```

        }
        Toast.makeText(getApplicationContext(), selectedSuperStar,
                Toast.LENGTH_LONG).show(); // print the value of selected super star
    });
}
}
}

```

Step 4: Open res -> values -> strings.xml

In this step we open String file which is used to store string data of the app.

```

<resources>
    <string name="app_name">RadioButtonExample</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="randyOrton">Randy Orton</string>
    <string name="johnCena">John Cena</string>
    <string name="romanReigns">Roman Reigns</string>
    <string name="goldBerg">Gold Berg</string>
    <string name="sheamus">Sheamus</string>
</resources>

```

Run The App:

Now run the App in Emulator and you will see 5 RadioButton in RadioGroup listing WWE superstar name. Now choose your favorite one and click on Submit Button. The name will be displayed on Screen.



CHAPTER 4

LAYOUTS AND RESOURCES

A container is a view used to contain other views. Android offers a collection of view classes that act as containers for views. These container classes are called layouts, and as the name suggests, they decide the organization, size, and position of their children views. Let's start the chapter with an introduction to different layouts used in Android applications.

4.1 INTRODUCTION TO LAYOUTS

Layouts are basically containers for other items known as Views, which are displayed on the screen. Layouts help manage and arrange views as well. Layouts are defined in the form of XML files that cannot be changed by our code during runtime.

The following Table shows the layout managers provided by the Android SDK.

Layout Manager	Description
LinearLayout	Organizes its children either horizontally or vertically
RelativeLayout	Organizes its children relative to one another or to the parent
AbsoluteLayout	Each child control is given a specific location within the bounds of the container
FrameLayout	Displays a single view; that is, the nextview replaces the previous view and hence is used to dynamically change the children in the layout
TableLayout	Organizes its children in tabular form
GridLayout	Organizes its children in grid format

The containers or layouts listed in Table 3.1 are also known as ViewGroups as one or more Views are grouped and arranged in a desired manner through them. Besides the ViewGroups shown

here Android supports one more ViewGroup known as ScrollView, which is discussed in Chapter 4, “Utilizing Resources and Media.”

4.2 LINEARLAYOUT

The LinearLayout is the most basic layout, and it arranges its elements sequentially, either horizontally or vertically. To arrange controls within a linear layout, the following attributes are used:

- ▶ **android:orientation**—Used for arranging the controls in the container in horizontal or vertical order
- ▶ **android:layout_width**—Used for defining the width of a control
- ▶ **android:layout_height**—Used for defining the height of a control
- ▶ **android:padding**—Used for increasing the whitespace between the boundaries of the control and its actual content
- ▶ **android:layout_weight**—Used for shrinking or expanding the size of the control to consume the extra space relative to the other controls in the container
- ▶ **android:gravity**—Used for aligning content within a control
- ▶ **android:layout_gravity**—Used for aligning the control within the container

Applying the orientation Attribute

The orientation attribute is used to arrange its children either in horizontal or vertical order. The valid values for this attribute are horizontal and vertical. If the value of the android:orientation attribute is set to vertical, the children in the linear layout are arranged in a column layout, one below the other. Similarly, if

the value of the android:orientation attribute is set to horizontal, the controls in the linear layout are arranged in a row format, side by side. The orientation can be modified at runtime through the setOrientation() method. That is, by supplying the values HORIZONTAL or VERTICAL to the setOrientation() method, we can arrange the children of the LinearLayout in row or column format, respectively.

Applying the height and widthAttributes

The default height and width of a control are decided on the basis of the text or content that is displayed through it. To specify a certain height and width to the control, we use the android:layout_width and android:layout_height attributes. We can specify the values for the height and width attributes in the following three ways:

- ▶ By supplying specific dimension values for the control in terms of px (pixels), dip/ dp (device independent pixels), sp (scaled pixels), pts (points), in (inches), and mm (millimeters). For example, the android:layout_width="20px" attribute sets the width of the control to 20 pixels.
- ▶ By providing the value as wrap_content. When assigned to the control's height or width, this attribute resizes the control to expand to fit its contents. For example, when this value is applied to the width of the TextView, it expands so that its complete text is visible.
- ▶ By providing the value as match_parent. When assigned to the control's height or width, this attribute forces the size of the control to expand to fill up all the available space of the enclosing container.

NOTE

For layout elements, the value `wrap_content` resizes the layout to fit the controls added as its children. The value `match_parent` makes the layout expand to take up all the space in the parent layout.

Applying the padding Attribute

The padding attribute is used to increase the whitespace between the boundaries of the control and its actual content. Through the `android:padding` attribute, we can set the same amount of padding or spacing on all four sides of the control. Similarly, by using the `android:paddingLeft`, `android:paddingRight`, `android:paddingTop`, and `android:paddingBottom` attributes, we can specify the individual spacing on the left, right, top, and bottom of the control, respectively.

The following example sets the spacing on all four sides of the control to 5 pixels:

```
android:padding="5dip"
```

Similarly, the following example sets the spacing on the left side of the control to 5 pixels:

```
android:paddingLeft="5dip"
```

NOTE

To set the padding at runtime, we can call the `setPadding()` method.

Let's see how the controls are laid out in the LinearLayout layout using an example. Create a new Android Project called LinearLayoutApp. The original default content of the layout file activity_linear_layout_app.xml appears as shown in Listing

**LISTING Default Code in the Layout File
activity_linear_layout_app.xml**

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"
        tools:context=".LinearLayoutAppActivity" />
</RelativeLayout>
```

Let's apply the LinearLayout and add three Button controls to the layout. Modify the activity_linear_layout_app.xml to appear as shown in Listing.

LISTING The activity_linear_layout_app.xml File on Adding Three ButtonControls

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The orientation of LinearLayout is set to vertical, declaring that we want to arrange its child elements vertically, one below the other. The height and width of the layout are set to expand to fill up all the available space of the enclosing container, that is, the device screen. Three Button controls are added to the layout, which appear one below the other. The IDs and text assigned to the three Button controls are Apple, Mango, and Banana, respectively. The height of the three controls is set to wrap_content, which is enough to accommodate the text. Finally, the width of the three controls

is set to `match_parent`, so that the width of the three controls expands to fill up the available space of the `LinearLayout` container. We see the output shown in Figure.



FIGURE Three Button controls arranged vertically in `LinearLayout`

To see the controls appear horizontally, set the `orientation` attribute of the `LinearLayout` to `horizontal`. We also need to set the `layout_width` attribute of the three controls to `wrap_content`; otherwise, we will be able to see only the first Button control, the one with the Apple ID. If the `layout_width` attribute of any control is set to `match_parent`, it takes up all the available space of the container, hiding the rest of the controls behind it. By setting the values of the `layout_width` attributes to `wrap_content`, we make sure that the width of the control expands just to fit its content and does not take up all the available space. Let's modify the `activity_linear_layout_app.xml` to appear as shown in Listing 3.3.

LISTING The `activity_linear_layout_app.xml` File on Setting Horizontal Orientation to the Button Controls

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="horizontal" >
```

```

<Button
    android:id="@+id/Apple"
    android:text="Apple"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<Button
    android:id="@+id/Mango"
    android:text="Mango"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<Button
    android:id="@+id/Banana"
    android:text="Banana"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>

```

The controls are arranged horizontally, as shown in Figure.

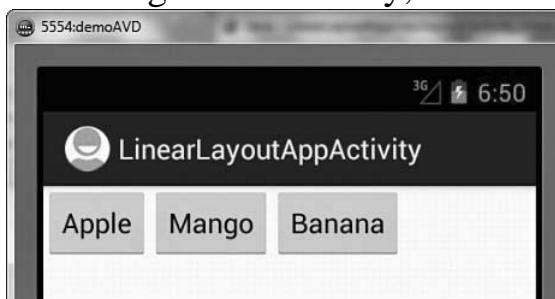


FIGURE Three Button controls arranged horizontally in LinearLayout

Applying the weight Attribute

The weight attribute affects the size of the control. That is, we use weight to assign the capability to expand or shrink and consume extra space relative to the other controls in the container. The values of the weight attribute range from 0.0 to 1.0, where 1.0 is the highest value. Let's suppose a container has two controls and one of them is assigned the weight of 1. In that case, the control

assigned the weight of 1 consumes all the empty space in the container, whereas the other control remains at its current size. If we assign a weight of 0.0 to both the controls, nothing happens and the controls maintain their original size. If both the attributes are assigned the same value above 0.0, both the controls consume the extra space equally. Hence, weight lets us apply a size expansion ratio to

the controls. To make the middle Button control, Mango, take up all the available space of the container, let's assign a weight attribute to the three controls. Modify the activity_linear_layout_app.xml file to appear as shown in Listing.

LISTING The activity_linear_layout_app.xml File on Applying the weightAttribute to the Button Controls

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="horizontal"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <Button  
        android:id="@+id/Apple"  
        android:text="Apple"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_weight="0.0" />  
    <Button  
        android:id="@+id/Mango"  
        android:text="Mango"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_weight="1.0" />  
    <Button  
        android:id="@+id/Banana"  
        android:text="Banana"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.0" />
</LinearLayout>
```

By setting the layout_weight attributes of Apple, Mango, and Banana to 0.0, 1.0, and 0.0, respectively, we allow the Mango button control to take up all the available space of the container, as shown in Figure (left). If we set the value of layout_weight of the Banana button control to 1.0 and that of Mango back to 0.0, then all the available space of the container is consumed by the Banana button control, as shown in Figure (middle).

Similarly if we set the layout_weight of all controls to 1.0, the entire container space will be equally consumed by the three controls, as shown in Figure (right).

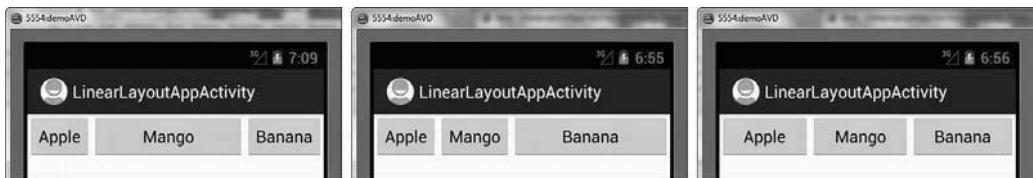


FIGURE (left) The weight attribute of the Mango Button control set to 1.0, (middle) the weight attribute of the Banana Button control set to 1.0, and (right) all three Button controls set to the same weight attribute

Similarly if we set the weight of Apple, Mango, and Banana to 0.0, 1.0, and 0.5, respectively, we get the output shown in Figure 3.4.

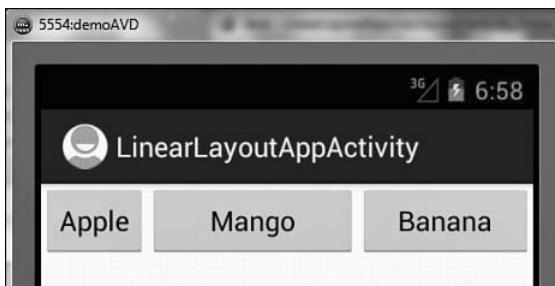


FIGURE The weight attribute of the Apple, Mango, and Banana Buttoncontrols set to 0.0, 1.0, and 0.5

We can see that the text of the three controls is center-aligned. To align the content of a control, we use the Gravity attribute.

Applying the Gravity Attribute

The Gravity attribute is for aligning the content within a control. For example, to align the text of a control to the center, we set the value of its android:gravity attribute to center. The valid options for android:gravity include left, center, right, top, bottom, center_horizontal, center_vertical, fill_horizontal, and fill_vertical. The task performed by few of the said options is as follows:

- ▶ **center_vertical**—Places the object in the vertical center of its container, without changing its size
- ▶ **fill_vertical**—Grows the vertical size of the object, if needed, so it completely fills its container
- ▶ **center_horizontal**—Places the object in the horizontal center of its container, without changing its size
- ▶ **fill_horizontal**—Grows the horizontal size of the object, if needed, so it completely fills its container

- **center**—Places the object in the center of its container in both the vertical and horizontal axis, without changing its size

We can make the text of a control appear at the center by using the android:gravity attribute, as shown in this example:

```
android:gravity="center"
```

We can also combine two or more values of any attribute using the | operator. The following example centrally aligns the text horizontally and vertically within a control:

```
android:gravity="center_horizontal|center_vertical"
```

Figure shows the android:gravity attribute set to left and right for the Button controls Mango and Banana.

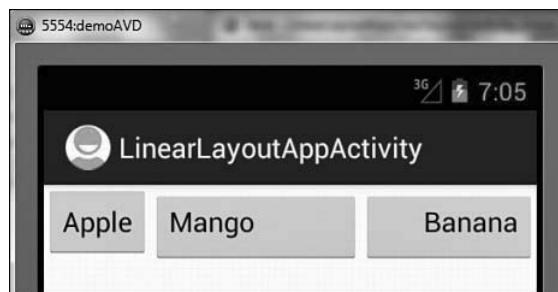


FIGURE The text in the Mango and Banana Button controls aligned to the left and right, respectively, through the android:gravity attribute

Besides the android:gravity attribute, Android provides one more similar attribute,

android:layout_gravity. Let's explore the difference between the two.

Using the android:layout_gravity Attribute

Where android:gravity is a setting used by the View, the

android:layout_gravity is used by the container. That is, this attribute is used to align the control within the container. For example, to align the text within a Button control, we use the android:gravity attribute; to align the Button control itself in the LinearLayout (the container), we use the android:layout_gravity attribute. Let's add the android:layout_gravity attribute to align the Button controls themselves. To see the impact of using the android:layout_gravity attribute to align the Button controls in the LinearLayout, let's first arrange them vertically. So, let's modify activity_linear_layout_app.xml to make the Button controls appear vertically, one below the other as shown in Listing.

LISTING expands to fill up the available space of the LinearLayout container. We see the output shown in Figure. The activity_linear_layout_app.xml File on Arranging the Button Controls Vertically

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The preceding code arranges the Button controls vertically, as shown in Figure 3.6 (left). To align the Button controls Mango and Banana to the center and to the right of the LinearLayout container, add the following statements to the respective tags in the activity_linear_layout_app.xml layout file:

```
        android:layout_gravity="center"           and
        android:layout_gravity="right"
```

The two Button controls, Mango and Banana, are aligned at the center and to the right in the container, as shown in Figure (middle).

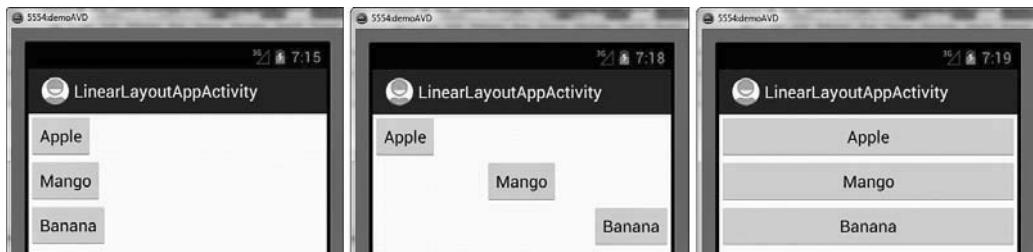


FIGURE (left) The three Button controls vertically aligned with the width attribute set to wrap_content, (middle) the Mango and Banana Button controls aligned to the center and right of container, and (right) the width of the three Button controls expanded to take up all the available space

At the moment, the layout_width attribute of the three controls is set to wrap_content. The width of the three controls is just enough to accommodate their content. If we now set the value of the android:layout_width attribute for all three controls to match_parent, we find that all three Button controls expand in width to take up all the available space of the container, as shown in Figure 3.6 (right). Now we can apply the android:gravity attribute to align the text within the controls.

Let's add the following three attributes to the Button controls Apple, Mango, and Banana:

```
android:gravity="left"  
android:gravity="center"  
and  
android:gravity="right"
```

These lines of code align the content of the three Button controls to the left, to the center, and to the right within the control, as shown in Figure (left). Because the three Button controls are arranged vertically in the layout (the orientation of the LinearLayout is set to vertical), the application of the weight attribute makes the controls expand vertically instead of horizontally as we saw earlier. To see the effect, let's add the following statement to the tags of all three Button controls:

```
android:layout_weight="0.0"
```

As expected, there will be no change in the height of any control, as the weight value assigned is 0.0. Setting an equal value above 0.0 for all three controls results in equal division of empty space among them. For example, assigning the android:layout_weight="1.0" to all three controls results in expanding their height, as shown in Figure

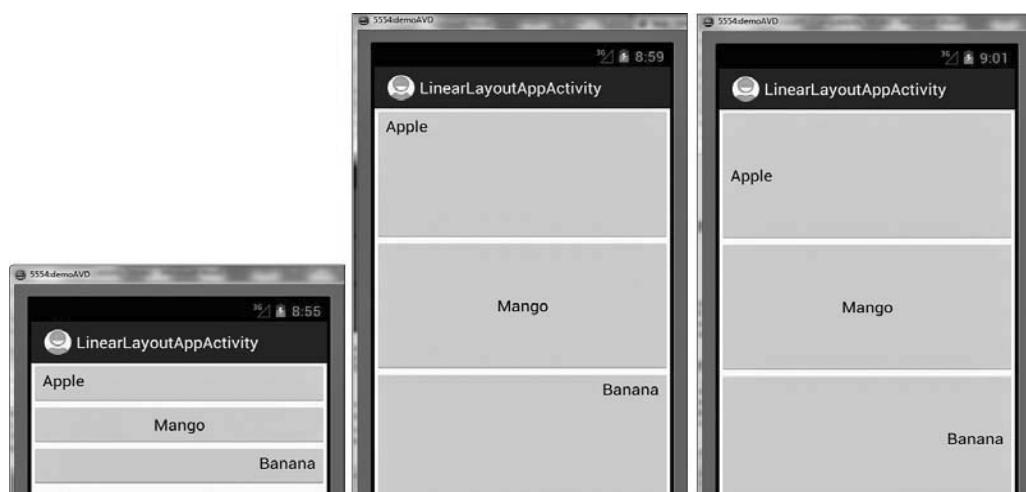


FIGURE (left) The three Button controls with their text aligned to the left, center, and right, (middle) the vertical available space of the container apportioned equally among the three Button controls, and (right) the text of the three Button controls vertically aligned to the center

In the middle image of Figure, we see that the text in the Apple and Banana controls is not at the vertical center, so let's modify their android:gravity value, as shown here:

android:gravity="center_vertical" for the Applecontrol

android:gravity="center_vertical|right" for the Banana control

The center_vertical value aligns the content vertically to the center of the control, and the right value aligns the content to the right of the control. We can combine the values of the attribute using the | operator. After applying the values as shown in the preceding two code lines, we get the output shown in Figure (right).

4.3 RELATIVE LAYOUT

In RelativeLayout, each child element is laid out in relation to other child elements; that is, the location of a child element is specified in terms of the desired distance from the existing children. To understand the concept of relative layout practically, let's create a new Android project called RelativeLayoutApp. Modify its layout file activity_relative_layout_app.xml to appear as shown in Listing.

LISTING The activity_relative_layout_app.xml File on Arranging the ButtonControls in the RelativeLayout Container

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="15dip"
        android:layout_marginLeft="20dip" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="28dip"
        android:layout_toRightOf="@+id/Apple"
        android:layout_marginLeft="15dip"
        android:layout_marginRight="10dip"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="200dip"
        android:layout_height="50dip"
        android:layout_marginTop="15dip"
        android:layout_below="@+id/Apple"
        android:layout_alignParentLeft="true" />
    <Button
        android:id="@+id/Grapes"
        android:text="Grapes"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:minWidth="100dp" />
```

```
    android:layout_alignParentRight="true"
    android:layout_below="@+id/Banana"
  />
<Button
    android:id="@+id/Kiwi"
    android:text="Kiwi"
    android:layout_width="100dip"
    android:layout_height="wrap_content"
    android:layout_below="@+id/Banana"
    android:paddingTop="15dip"
    android:paddingLeft="25dip"
    android:paddingRight="25dip" />
</RelativeLayout>
```

Before we understand how the controls in the previous code block are placed, let's have a quick look at different attributes used to set the positions of the layout controls.

Layout Control Attributes

The attributes used to set the location of the control relative to a container are

- ▶ **android:layout_alignParentTop**—The top of the control is set to align with the top of the container.
- ▶ **android:layout_alignParentBottom**—The bottom of the control is set to align with the bottom of the container.
- ▶ **android:layout_alignParentLeft**—The left side of the control is set to align with the left side of the container.
- ▶ **android:layout_alignParentRight**—The right side of the control is set to align with the right side of the container.
- ▶ **android:layout_centerHorizontal**—The control is placed horizontally at the center of the container.
- ▶ **android:layout_centerVertical**—The control is placed

vertically at the center of the container.

- ▶ **android:layout_centerInParent**—The control is placed horizontally and vertically at the center of the container.

The attributes to control the position of a control in relation to other controls are

- ▶ **android:layout_above**—The control is placed above the referenced control.
- ▶ **android:layout_below**—The control is placed below the referenced control.
- ▶ **android:layout_toLeftOf**—The control is placed to the left of the referenced control.
- ▶ **android:layout_toRightOf**—The control is placed to the right of the referenced control.

The attributes that control the alignment of a control in relation to other controls are

- ▶ **android:layout_alignTop**— The top of the control is set to align with the top of the referenced control.
- ▶ **android:layout_alignBottom**—The bottom of the control is set to align with the bottom of the referenced control.
- ▶ **android:layout_alignLeft**—The left side of the control is set to align with the left side of the referenced control.
- ▶ **android:layout_alignRight**—The right side of the control is set to align with the right side of the referenced control.
- ▶ **android:layout_alignBaseline**—The baseline of the two controls will be aligned.

For spacing, Android defines two attributes: `android:layout_margin`

and `android:padding`. The `android:layout_margin` attribute defines spacing for the container, while `android:padding` defines the spacing for the view. Let's begin with padding.

- ▶ **android:padding**—Defines the spacing of the content on all four sides of the control. To define padding for each side individually, use `android:paddingLeft`, `android:paddingRight`, `android:paddingTop`, and `android:paddingBottom`.
- ▶ **android:paddingTop**—Defines the spacing between the content and the top of the control.
- ▶ **android:paddingBottom**—Defines the spacing between the content and the bottom of the control.
- ▶ **android:paddingLeft**—Defines the spacing between the content and the left side of the control.
- ▶ **android:paddingRight**—Defines the spacing between the content and the right side of the control.

Here are the attributes that define the spacing between the control and the container:

- ▶ **android:layout_margin**—Defines the spacing of the control in relation to the controls or the container on all four sides. To define spacing for each side individually, we use the `android:layout_marginLeft`, `android:layout_marginRight`, `android:layout_marginTop`, and `android:layout_marginBottom` options.
- ▶ **android:layout_marginTop**—Defines the spacing between the top of the control and the related control or container.
- ▶ **android:layout_marginBottom**—Defines the spacing between the bottom of the control and the related control or container.

- ▶ **android:layout_marginRight**—Defines the spacing between the right side of the control and the related control or container.
- ▶ **android:layout_marginLeft**—Defines the spacing between the left side of the control and the related control or container.

The layout file activity_relative_layout_app.xml arranges the controls as follows:

The Apple button control is set to appear at a distance of 15dip from the top and 20dip from the left side of the RelativeLayout container. The width of the Mango button control is set to consume the available horizontal space. The text Mango appears at a distance of 28dip from all sides of the control. The Mango control is set to appear to the right of the Apple control. The control is set to appear at a distance of 15dip from the control on the left and 10dip from the right side of the relative

layout container. Also, the top of the Button control is set to align with the top of the container.

The Banana button control is assigned the width and height of 200dip and 50dip, respectively. The control is set to appear 15dip below the Apple control. The left side of the control is set to align with the left side of the container.

The Grapes button control is set to appear below the Banana button control, and its width is set to expand just enough to accommodate its content. The height of the control is set to take up all available vertical space. The text Grapes is automatically aligned vertically; that is, it appears at the center of the vertical height when the height attribute is set to match_parent. The minimum width of the control is set to 100dip. The right side of the control is set to align with the right side of the container.

The Kiwi Button control is set to appear below the Banana control. Its width is set to 100dip, and the height is set to just accommodate its content. The text Kiwi is set to appear at the distance of 15dip, 25dip, and 25dip from the top, left, and right boundary of the control.

We don't need to make any changes to the RelativeLayoutAppActivity.java file. Its original content is as shown in Listing.

LISTING The Default Code in the Activity File

```
RelativeLayoutAppActivity.java package  
com.androidunleashed.relativelayoutapp;  
  
import android.app.Activity;  
import android.os.Bundle;  
  
public class RelativeLayoutDemoActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_relative_layout_app);  
    }  
}
```

When the application is run, we see the output shown in Figure

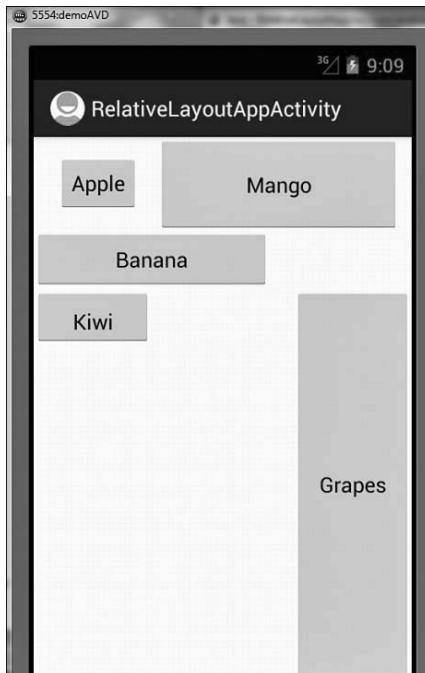


FIGURE The five Button controls' layout relative to each other

We can make the text Gapes appear centrally at the top row by adding the following line:

```
android:gravity="center_horizontal"
```

So, its tag appears as follows:

```
<Button  
    android:id="@+id/Grapes"  
    android:text="Grapes"  
    android:layout_width="wrap_content"  
    android:layout_height="match_parent"  
    android:minWidth="100dp"  
    android:layout_alignParentRight="true"  
    android:layout_below="@+id/Banana"  
    android:gravity="center_horizontal" />
```

The output is modified to appear as shown in Figure

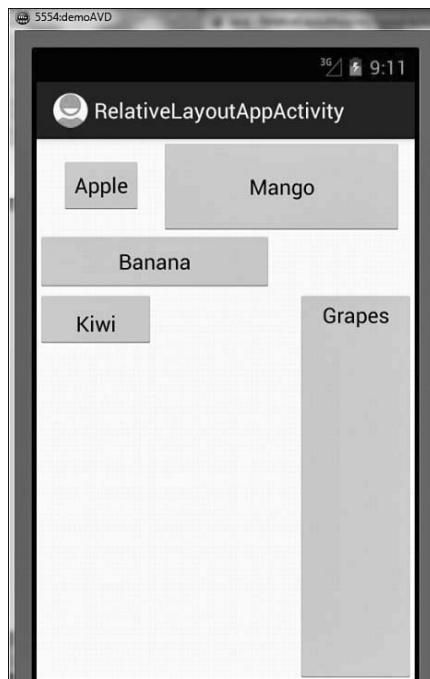


FIGURE The Grapes Button control aligned horizontally at the center

Let's explore the concept of laying out controls in the `RelativeLayout` container by writing an application. The application that we are going to create is a simple Login Form application that asks the user to enter a User ID and Password. The `TextView`, `EditText`, and `Button` controls in the application are laid out in a `RelativeLayout` container (see Figure 3.10—left). If either the User ID or Password is left blank, the message The User ID or password is left blank. Please Try Again is displayed. If the correct User ID and Password, in this case, guest, are entered, then a welcome message is displayed. Otherwise, the message The User ID or password is incorrect. Please Try Again is displayed.

So, let's create the application. Launch the Eclipse IDE and create a new Android application called `LoginForm`. Arrange four `TextView` controls, two `EditText` controls, and a `Button` control in `RelativeLayout`, as shown in the layout file `activity_login_form.xml`.

displayed in Listing.

LISTING The activity_login_form.xml on Laying Out the TextView, EditText, and Button Controls in the RelativeLayout Container

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/sign_msg"
        android:text="Sign In"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:typeface="serif"    android:textSize="25dip"
        android:textStyle="bold"    android:padding="10dip"
        android:layout_centerHorizontal="true"/>
    <TextView
        android:id="@+id/user_msg"    android:text="User ID:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dip"
        android:layout_below="@+id/sign_msg" />
    <EditText
        android:id="@+id/user_ID"
        android:layout_height="wrap_content"
        android:layout_width="250dip"
        android:layout_below="@+id/sign_msg"
```

```
    android:layout_toRightOf="@+id/user_msg"
    android:singleLine="true" />
<TextView
    android:id="@+id/password_msg"
    android:text = "Password:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/user_msg"
    android:layout_margin="10dip" android:paddingTop="10dip"/>
<EditText
    android:id="@+id/password"
    android:layout_height="wrap_content"
    android:layout_width="250dp"
    android:singleLine="true"
    android:layout_below="@+id/user_ID"
    android:layout_toRightOf="@+id/password_msg"
    android:password="true" />
<Button
    android:id="@+id/login_button" android:text="Sign In"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dip"
    android:layout_below="@+id/password_msg"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/response"
```

```
    android:layout_below="@+id/login_button"/>  
</RelativeLayout>
```

The controls in the application are arranged in the RelativeLayout, as explained here:

- ▶ Through the TextView control sign_msg, the text Sign In is displayed horizontally centered at the top. It is displayed in bold serif font, 25 dip in size. The text is padded with a space of 10dip on all four sides of its container.
- ▶ Another TextView control, user_msg, displays the text User ID below the TextView sign_msg. The TextView is placed 10dip from all four sides.
- ▶ An EditText control user_ID is displayed below sign_msg and to the right of user_ msg. The width assigned to the TextView control is 250 dip and is set to single-line mode, so if the user types beyond the given width, the text scrolls to accommodate extra text but does not run over to the second line.
- ▶ A TextView password_msg control displaying the text Password: is displayed below the TextView user_msg. The TextView control is placed at a spacing of 10dip from all four sides, and the text Password: is displayed at 10dip from the control's top boundary.
- ▶ An EditTextcontrol passwordis displayed below the EditText user_ID and to the right of the TextView password_msg. The widthassigned to the TextViewcontrol is 250 dip and is set to single-line mode. In addition, the typed characters are converted into dots for security.
- ▶ A Button control login_button with the caption Sign In is displayed below the TextView password_msg. The button is

horizontally centered and is set to appear at 10dip distance from the EditText control password.

- A TextView control response is placed below the Button login_button. It is used to display messages to the user when the Sign In button is pressed after entering User ID and Password.

To authenticate the user, we need to access the User ID and Password that is entered and match these values against the valid User ID and Password. In addition, we want to validate the EditText controls to confirm that none of them is blank. We also want to welcome the user if he or she is authorized. To do all this, we write the code in the activity file LoginFormActivity.java as shown in Listing.

LISTING Code Written in the Java Activity File LoginFormActivity.java

```
package com.androidunleashed.loginform;  
  
Import android.app.Activity;  
import android.os.Bundle;  
import android.view.View.OnClickListener;  
import android.widget.Button;  
import android.widget.EditText;  
import android.view.View;  
import android.widget.TextView;  
  
public class LoginActivity extends Activity implements  
    OnClickListener { @Override  
    public void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_login_form);
Button b = (Button)this.findViewById(R.id.login_button);
b.setOnClickListener(this);
}

public void onClick(View v) {
    EditText userid = (EditText) findViewById(R.id.user_ID);
    EditText password = (EditText) findViewById(R.id.password);
    TextView resp = (TextView)this.findViewById(R.id.response);
    String usr = userid.getText().toString();
    String pswd = password.getText().toString();
    if(usr.trim().length()==0||pswd.trim().length()==0){
        String str = "The User ID or password is left blank \nPlease
        Try Again"; resp.setText(str);
    }
    else{
        if(usr.equals("guest") && pswd.equals("guest"))
            resp.setText("Welcome " + usr+ " !");
        else resp.setText("The UserID or password is incorrect\nPlease
        Try Again");
    }
}
}

```

The Button control is accessed from the layout file and is mapped to the Button object b. This activity implements the onClickListener interface. Hence, the class implements the callback method onClick(), which is invoked when a click event

occurs on the Button control.

In the onClick() method, the user_ID and password EditText controls are accessed from the layout file and mapped to the EditText objects userid and password. Also, the TextView control response is accessed from the layout file and is mapped to the TextView object resp. The User ID and password entered by the user in the two EditText controls are accessed through the objects userid and password and assigned to the two Strings usr and pswd, respectively. The data in the usr and pswd strings is checked for authentication. If the user has left any of the EditText controls blank, the message The User ID or password is left blank. Please Try Again is displayed, as shown in Figure (left). If the User ID and password are correct, then a welcome message is displayed (see Figure—right). Otherwise, the message The User ID or password is incorrect. Please Try Again is displayed, as shown in Figure (middle).



FIGURE (left) The Login Form displays an error if fields are left blank, (middle) the Password Incorrect message displays if the user ID or password is incorrect, and (right) the Welcome message displays when the correct user ID and password are entered.

4.4 ABSOLUTE LAYOUT

Each child in an `AbsoluteLayout` is given a specific location within the bounds of the container. Such fixed locations make `AbsoluteLayout` incompatible with devices of different screen size and resolution. The controls in `AbsoluteLayout` are laid out by specifying their exact *X* and *Y* positions. The coordinate 0,0 is the origin and is located at the top-left corner of the screen.

Let's write an application to see how controls are positioned in `AbsoluteLayout`. Create a new Android Project called `AbsoluteLayoutApp`. Modify its layout file, `activity_absolute_layout_app.xml`, as shown in Listing.

LISTING The Layout File `activity_absolute_layout_app.xml` on Arranging Controls in the `AbsoluteLayout` Container

```
<AbsoluteLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Product Form"
        android:textSize="20sp"
        android.textStyle="bold"
        android:layout_x="90dip" android:layout_y="2dip"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:text="Product Code:"  
    android:layout_x="5dip" android:layout_y="40dip" />  
<EditText  
    android:id="@+id/product_code"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:minWidth="100dip"  
    android:layout_x="110dip" android:layout_y="30dip"  
    />  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Product Name:"  
    android:layout_x="5dip" android:layout_y="90dip"/>  
<EditText  
    android:id="@+id/product_name"  
    android:layout_width="200dip"  
    android:layout_height="wrap_content"  
    android:minWidth="200dip"  
    android:layout_x="110dip" android:layout_y="80dip"  
    android:scrollHorizontally="true" />  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Product Price:"  
    android:layout_x="5dip" android:layout_y="140dip"  
    />  
<EditText
```

```
    android:id="@+id/product_price"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minWidth="100dip"
    android:layout_x="110dip"
    android:layout_y="130dip" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:id="@+id/click_btn"
    android:text="AddNewProduct"
    android:layout_x="80dip"
    android:layout_y="190dip" />

</AbsoluteLayout>
```

The controls in activity_absolute_layout_app.xml are as follows:

- ▶ The New Product Form TextView is set to appear 90dip from the left and 2dip from the top side of the container. The size of the text is set to 20sp, and its style is set to bold.
- ▶ The Product Code TextView is set to appear 5dip from the left and 40dip from the top side of the container.
- ▶ The product_code EditText control is set to appear 110dip from the left and 30dip from the top side of the container. The minimum width of the control is set to 100dp.
- ▶ The ProductName TextView control is set to appear 5dip from the left and 90dip from the top side of the container.
- ▶ The product_name EditText control is set to appear 110dip

from the left and 80dip from the top side of the container. The minimum width of the control is set to 200dip, and its text is set to scroll horizontally when the user types beyond its width.

- ▶ The Product Price TextView is set to appear 5dip from the left and 140dip from the top side of the container.
- ▶ The product_price EditText control is set to appear 110dip from the left and 130dip from the top side of the container. The minimum width of the control is set to 100dip.
- ▶ The click_btn Button, Add New Product, is set to appear 80dip from the left and 190dip from the top side of the container.

If we don't specify the x, y coordinates of a control in AbsoluteLayout, it is placed in the origin point, that is, at location 0,0. If the value of the x and y coordinates is too large, the control does not appear on the screen. The values of the x and y coordinates are specified in any units, such as sp, in, mm, and pt.

After specifying the locations of controls in the layout file activity_absolute_layout_.app.xml, we can run the application. There is no need to make any changes in the file AbsoluteLayoutAppActivity.java. When the application is run, we get the output shown in Figure.

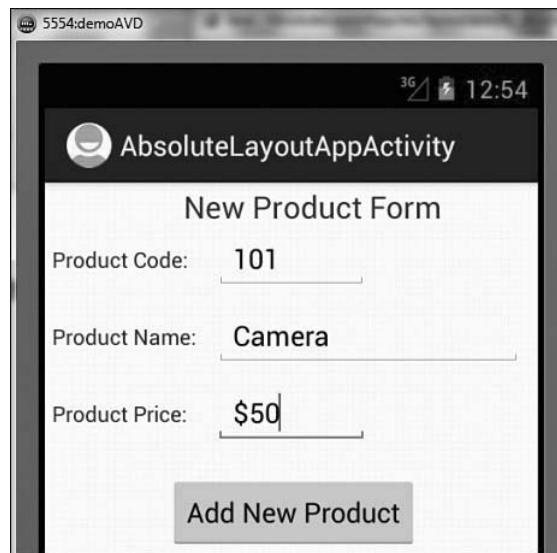


FIGURE Different controls laid out in AbsoluteLayout

The `AbsoluteLayout` class is not used often, as it is not compatible with Android phones of different screen sizes and resolutions.

The next layout we are going to discuss is `FrameLayout`. Because we will learn to display images in `FrameLayout`, let's first take a look at the `ImageView` control that is often used to display images in Android applications.

4.5 USING IMAGEVIEW

An `ImageView` control is used to display images in Android applications. An image can be displayed by assigning it to the `ImageView` control and including the `android:src` attribute in the XML definition of the control. Images can also be dynamically assigned to the `ImageView` control through Java code.

A sample `ImageView` tag when used in the layout file is shown here:

```
<ImageView  
    android:id="@+id/first_image"
```

```
    android:src = "@drawable/bintupic"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitXY"
    android:adjustViewBounds="true"
    android:maxHeight="100dip"
    android:maxLength="250dip"
    android:minHeight="100dip"
    android:minWidth="250dip"
    android:resizeMode="horizontal|vertical" />
```

Almost all attributes that we see in this XML definition should be familiar, with the exception of the following ones:

- ▶ **android:src**—Used to assign the image from drawable resources. We discuss drawable resources in detail in Chapter 4. For now, assume that the image in the res/ drawable folder is set to display through the ImageView control via this attribute.

Example:

```
    android:src = "@drawable/bintupic"
```

You do not need to specify the image file extension. JPG and GIF files are supported, but the preferred image format is PNG.

- ▶ **android:scaleType**—Used to scale an image to fit its container. The valid values for this attribute include fitXY, center, centerInside, and fitCenter. The value fitXY independently scales the image around the X and Y axes without maintaining the aspect ratio to match the size of container. The value center centers the image in the container without scaling it. The value centerInside scales the

image uniformly, maintaining the aspect ratio so that the width and height of the image fit the size of its container. The value fitCenter scales the image while maintaining the aspect ratio, so that one of its X or Y axes fits the container.

- ▶ **android:adjustViewBounds**—If set to true, the attribute adjusts the bounds of the ImageView control to maintain the aspect ratio of the image displayed through it.
- ▶ **android:resizeMode**—The resizeMode attribute is used to make a control resizable so we can resize it horizontally, vertically, or around both axes. We need to click and hold the control to display its resize handles. The resize handles can be dragged in the desired direction to resize the control. The available values for the resizeMode attribute include horizontal, vertical, and none. The horizontal value resizes the control around the horizontal axis, the vertical value resizes around the vertical axis, the both value resizes around both the horizontal and vertical axes, and the value none prevents resizing.

4.6 FRAMELAYOUT

FrameLayout is used to display a single View. The View added to a FrameLayout is placed at the top-left edge of the layout. Any other View added to the FrameLayout overlaps the previous View; that is, each View stacks on top of the previous one. Let's create an application to see how controls can be laid out using FrameLayout.

In the application we are going to create, we will place two ImageView controls in the FrameLayout container. As expected, only one ImageView will be visible, as one ImageView will overlap the other ImageView, assuming both ImageView controls are of the

same size. We will also display a button on the ImageView, which, when selected, displays the hidden ImageView underneath.

Let's start with the application. Create a new Android project called FrameLayoutApp.

To display images in Android applications, the image is first copied into the res/drawable folder and from there, it is referred to in the layout and other XML files. We look at the procedure for displaying images, as well as the concept of drawable resources, in detail in Chapter 4. For the time being, it is enough to know that to enable the image(s) to be referred to in the layout files placed in the res/drawable folder, the image needs to exist in the res/drawable folder. There are four types of drawable folders: drawable-xhdpi, drawable-hdpi, /res/drawable-mdpi, and /res/drawable-ldpi. We have to place images of different resolutions and sizes in these folders. The graphics with the resolutions 320 dpi, 240dpi, 160 dpi, and 120dpi (96 x 96 px, 72 x 72 px, 48 x 48 px, and 36 x 36 px), are stored in the res/drawable-xhdpi, res/drawable-hdpi, res/drawable-mdpi, and res/ drawable-ldpi folders, respectively. The application picks up the appropriate graphic from the correct folder. So, if we copy two images called bintupic.png and bintupic2.png of the preceding size and resolution and paste them into the four res/drawable folders, the Package Explorer resembles Figure.

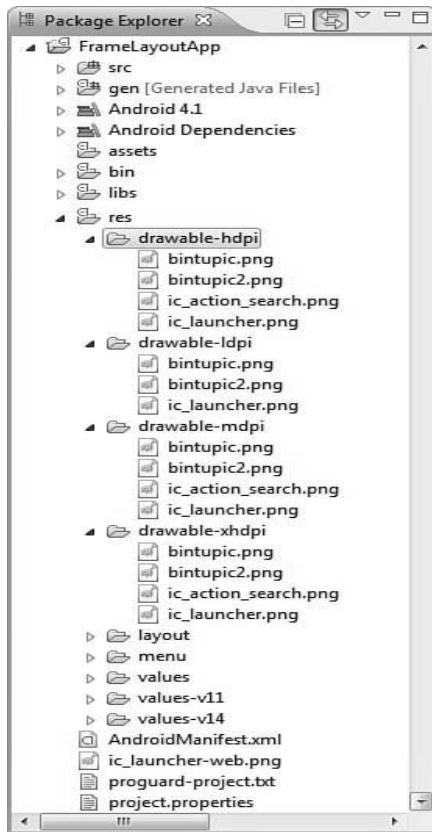


FIGURE The Package Explorer window showing the two images, bintupic.png and bintupic2.png, dropped into the res/drawable folders

To display two ImageViews and a TextView in the application, let's write the code in the layout file activity_frame_layout_app.xml as shown in Listing.

LISTING The Layout File activity_frame_layout_app.xml on Arranging the ImageView and TextView Controls in the FrameLayout Container

<FrameLayout

```
xmlns:android="http://schemas.android.com/apk/res/android"  
android:orientation="vertical"  
android:layout_width="match_parent"  
android:layout_height="match_parent">
```

```
<ImageView  
    android:id="@+id/first_image"  
    android:src = "@drawable/bintupic"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:scaleType="fitXY" />  
  
<ImageView  
    android:id="@+id/second_image"  
    android:src="@drawable/bintupic2"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:scaleType="fitXY" />  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Click the image to switch"  
    android:layout_gravity="center_horizontal|bottom"  
    android:padding="5dip" android:textColor="#ffffffff"  
    android:textStyle="bold"  
    android:background="#333333"  
    android:layout_marginBottom="10dip" />  
  
</FrameLayout>
```

The first_image and second_image ImageView controls are set to display the images bintupic.png and bintupic2.png, respectively. To make the two images stretch to cover the entire screen, the scaleType attribute in the ImageView tag is set to fitXY. A TextView, Click the image to switch, is set to display at the horizontally centered position and at a distance of 10dip from

the bottom of the container. The spacing between the text and the boundary of the TextView control is set to 5dip. The background of the text is set to a dark color, the foreground color is set to white, and its style is set to bold. When a user selects the current image on the screen, the image should switch to show the hidden image. For this to occur, we need to write code in the activity file as shown in Listing.

LISTING Code Written in the Java Activity File

FrameLayoutAppActivity.java

```
package com.androidunleashed.framelayoutapp;  
import android.app.Activity;  
import android.os.Bundle;  
import android.widget.ImageView;  
import android.view.View.OnClickListener;  
import android.view.View;  
  
public class FrameLayoutAppActivity extends Activity { @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_frame_layout_app);  
        final ImageView first_image =  
            (ImageView)this.findViewById(R.id.first_image);  
        final ImageView second_image =  
            (ImageView)this.findViewById(R.id.second_image);  
        first_image.setOnClickListener(new OnClickListener(){  
            public void onClick(View view) {  
                second_image.setVisibility(View.VISIBLE);  
            }  
        });  
    }  
}
```

```
        view.setVisibility(View.GONE);
    }
});

second_image.setOnClickListener(new OnClickListener(){
    public void onClick(View view) {
        first_image.setVisibility(View.VISIBLE);
        view.setVisibility(View.GONE);
    }
});
}
}
```

The two `first_image` and `second_image` `ImageView` controls are located through the `findViewById` method of the `Activity` class and assigned to the two `ImageView` objects, `first_image` and `second_image`, respectively. We register the click event by calling the `setOnClickListener()` method with an `OnClickListener`. An anonymous listener is created on the fly to handle click events for the `ImageView`. When the `ImageView` is clicked, the `onClick()` method of the listener is called. In the `onClick()` method, we switch the images; that is, we make the current `ImageView` invisible and the hidden `ImageView` visible. When the application runs, we see the output shown in Figure (left). The application shows an image, and the other image is hidden behind it because in `FrameLayout` one View overlaps the other. When the user clicks the image, the images are switched, as shown in Figure (right).

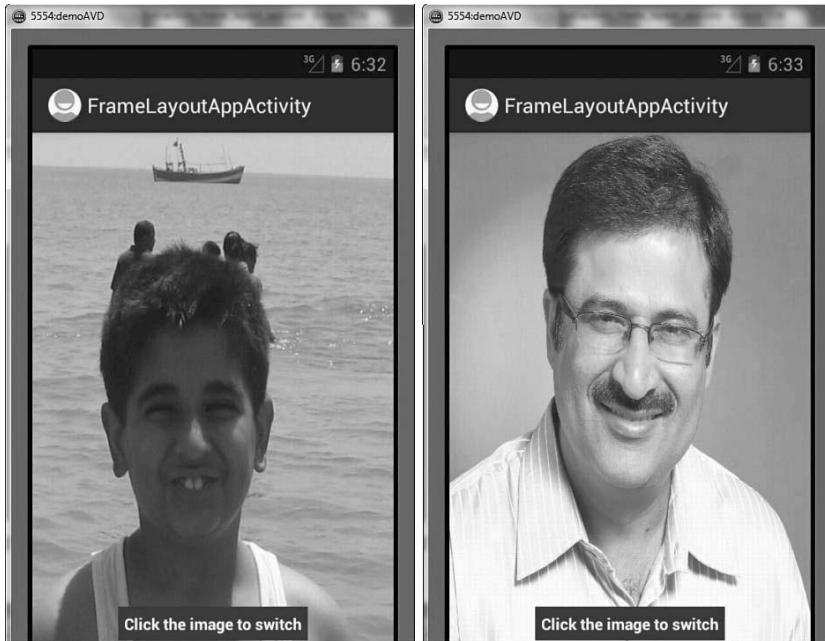


FIGURE (left) An image and a TextView laid out in FrameLayout, and (right) the images switch when clicked

4.7 TABLE LAYOUT

The TableLayout is used for arranging the enclosed controls into rows and columns. Each new row in the TableLayout is defined through a TableRow object. A row can have zero or more controls, where each control is called a cell. The number of columns in a TableLayout is determined by the maximum number of cells in any row. The width of a column is equal to the widest cell in that column. All elements are aligned in a column;

that is, the width of all the controls increases if the width of any control in the column is increased.

NOTE

We can nest another TableLayout within a table cell, as well.

Operations Applicable to TableLayout

We can perform several operations on TableLayout columns, including stretching, shrinking, collapsing, and spanning columns.

Stretching Columns

The default width of a column is set equal to the width of the widest column, but we can stretch the column(s) to take up available free space using the **android:stretchColumns attribute in the TableLayout**. The value assigned to this attribute can be a single column number or a comma-delimited list of column numbers. The specified columns are stretched to take up any available space on the row.

Examples:

- ▶ **android:stretchColumns="1"**—The second column (because the column numbers are zero-based) is stretched to take up any available space in the row.
- ▶ **android:stretchColumns="0,1"**—Both the first and second columns are stretched to take up the available space in the row.
- ▶ **android:stretchColumns="*"**—All columns are stretched to take up the available space.

Shrinking Columns

We can shrink or reduce the width of the column(s) using the **android:shrinkColumns attribute in the TableLayout**. We can specify either a single column or a comma-delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

NOTE

By default, the controls are not word-wrapped.

Examples:

- ▶ **android:shrinkColumns="0"**—The first column's width shrinks or reduces by word-wrapping its content.
- ▶ **android:shrinkColumns="*"**—The content of all columns is word-wrapped to shrink their widths.

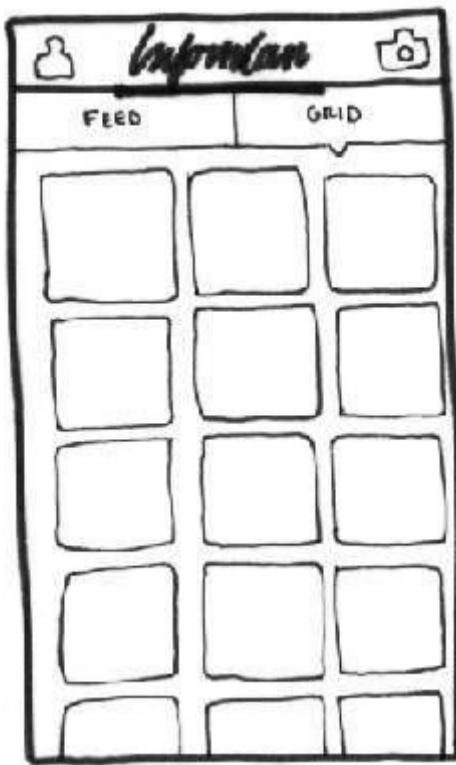
Collapsing Columns

We can make the column(s) collapse or become invisible through the android: collapseColumns attribute in the TableLayout. We can specify one or more comma-delimited columns for this attribute. These columns are part of the table information but are invisible. We can also make column(s) visible and invisible through coding by passing the Boolean values false and true, respectively, to the setColumnCollapsed() method in the TableLayout. For example:

- ▶ **android:collapseColumns="0"**—The first column appears collapsed; that is, it is part of the table but is invisible. It can be made visible through coding by using the setColumnCollapsed() method.

4.8 GRIDLAYOUT

Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter**



An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

GridView Attributes

Following are the important attributes specific to GridView –

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:columnWidth This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.

3	<p>android:gravity</p> <p>Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.</p>
4	<p>android:horizontalSpacing</p> <p>Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.</p>
5	<p>android:numColumns</p> <p>Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.</p>
6	<p>android:stretchMode</p> <p>Defines how columns should stretch to fill the available empty space, if any. This must be either of the values –</p> <ul style="list-style-type: none"> • none – Stretching is disabled. • spacingWidth – The spacing between each column is stretched. • columnWidth – Each column is stretched equally. • spacingWidthUniform – The spacing between each column is uniformly stretched..
7	<p>android:verticalSpacing</p> <p>Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.</p>

Example

This example will take you through simple steps to show how to create your own Android application using GridView. Follow the following steps to modify the Android application we created in *Hello World Example* chapter –

Step	Description
------	-------------

1	You will use Android studio IDE to create an Android application and name it as <i>HelloWorld</i> under a package <i>com.example.helloworld</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify the default content of <i>res/layout/activity_main.xml</i> file to include <i>GridView</i> content with the self explanatory attributes.
3	No need to change <i>string.xml</i> , Android studio takes care of defaults strings which are placed at <i>string.xml</i>
4	Let's put few pictures in <i>res/drawable-hdpi</i> folder. I have put <i>sample0.jpg</i> , <i>sample1.jpg</i> , <i>sample2.jpg</i> , <i>sample3.jpg</i> , <i>sample4.jpg</i> , <i>sample5.jpg</i> , <i>sample6.jpg</i> and <i>sample7.jpg</i> .
5	Create a new class called ImageAdapter under a package <i>com.example.helloworld</i> that extends <i>BaseAdapter</i> . This class will implement functionality of an adapter to be used to fill the view.
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.helloworld/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.helloworld;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.GridView;
```

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
        GridView gridview = (GridView) findViewById(R.id.gridView);
        gridview.setAdapter(new ImageAdapter(this));
    }
}
```

Following will be the content of **res/layout/activity_main.xml** file –

```
<?xml version="1.0" encoding="utf-8"?>
<GridView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

Following will be the content of **res/values/strings.xml** to define two new constants

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="action_settings">Settings</string>
</resources>
```

Following will be the content of **src/com.example.helloworld/ImageAdapter.java** file –

```
package com.example.helloworld;
```

```
import android.content.Context;
```

```
import android.view.View;
import android.view.ViewGroup;

import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    // Constructor
    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;

        if (convertView == null) {
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85,
85));
        }
    }
}
```

```

imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
    imageView.setPadding(8, 8, 8, 8);
}
else
{
    imageView = (ImageView) convertView;
}
imageView.setImageResource(mThumbIds[position]);
return imageView;
}

// Keep all Images in array
public Integer[] mThumbIds = {
    R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5,
    R.drawable.sample_6, R.drawable.sample_7,
    R.drawable.sample_0, R.drawable.sample_1,
    R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5,
    R.drawable.sample_6, R.drawable.sample_7,
    R.drawable.sample_0, R.drawable.sample_1,
    R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5,
    R.drawable.sample_6, R.drawable.sample_7
};
}

```

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window –



4.9 ADAPTING TO SCREEN ORIENTATION

The `screenOrientation` is the attribute of `activity` element. The orientation of android activity can be portrait, landscape, sensor, unspecified etc. You need to define it in the `AndroidManifest.xml` file.

Syntax:

```
<activity android:name="package_name.Your_ActivityName"  
        android:screenOrientation="orirntation_type">  
</activity>
```

Example:

```
<activity android:name="  
example.javatpoint.com.screenorientation.MainActivity"  
        android:screenOrientation="portrait">  
</activity>  
<activity android:name=".SecondActivity"  
        android:screenOrientation="landscape">  
</activity>
```

The common values for `screenOrientation` attribute are as follows:

Value Description

`unspecified` It is the default value. In such case, system chooses the

orientation.

portrait taller not wider

landscape wider not taller

sensor orientation is determined by the device orientation sensor.

Android Portrait and Landscape mode screen orientation example

In this example, we will create two activities of different screen orientation. The first activity (MainActivity) will be as "portrait" orientation and second activity (SecondActivity) as "landscape" orientation type.

activity_main.xml

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    tools:context="example.javatpoint.com.screenorientation.MainActivity"
>

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginTop="112dp"
    android:onClick="onClick"
    android:text="Launch next activity"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.612"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editText1"
```

```
    app:layout_constraintVertical_bias="0.613" />

<TextView
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="124dp"
    android:ems="10"
    android:textSize="22dp"
    android:text="This activity is portrait orientation"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.502"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Activity class

File: MainActivity.java

```
package example.javatpoint.com.screenorientation;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    Button button1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

    button1=(Button)findViewById(R.id.button1);
}
public void onClick(View v) {
    Intent intent = new
Intent(MainActivity.this,SecondActivity.class);
    startActivity(intent);
}
}
```

activity_second.xml

File: activity_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    tools:context="example.javatpoint.com.screenorientation.SecondActi
    vity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="180dp"
        android:text="this is landscape orientation"
        android:textSize="22dp"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.502"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

SecondActivity class

File: SecondActivity.java

```
package example.javatpoint.com.screenorientation;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);

    }
}
```

AndroidManifest.xml

File: AndroidManifest.xml

In AndroidManifest.xml file add the screenOrientation attribute in activity and provides its orientation. In this example, we provide "portrait" orientation for MainActivity and "landscape" for SecondActivity.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
        package="example.javatpoint.com.screenorientation">

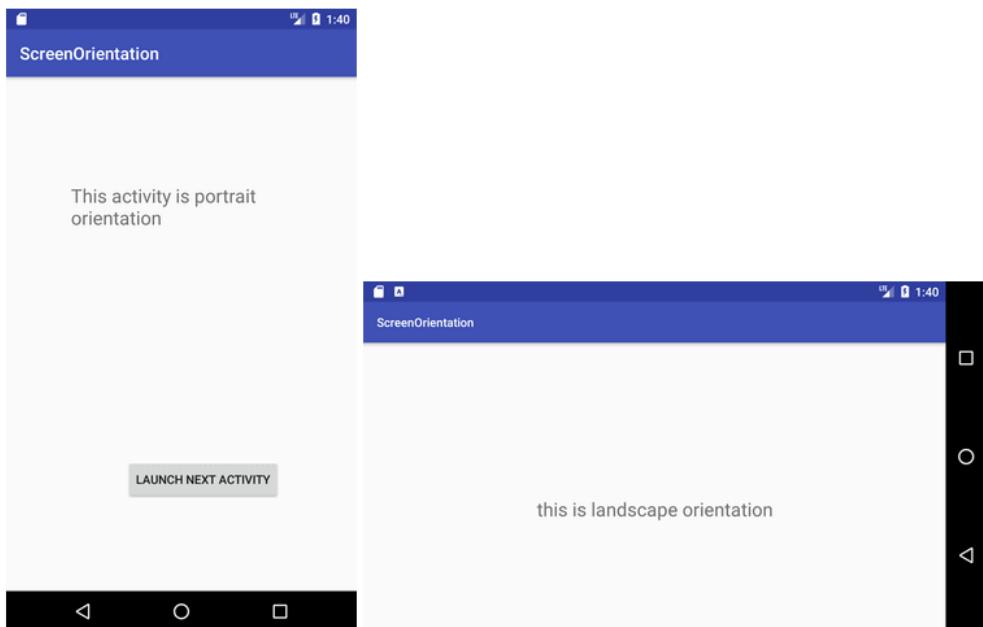
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity

            android:name="example.javatpoint.com.screenorientation.MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"
            android:screenOrientation="landscape">
        </activity>
    </application>

</manifest>
```

Output:



4.10 CREATING VALUES RESOURCES USING DRAWABLE RESOURCES

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other **resources** like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.

This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications.

Organize resource in Android Studio

```

MyProject/
  app/
    manifest/
      AndroidManifest.xml
  java/
    MyActivity.java
  res/
    drawable/
      icon.png
    layout/
      activity_main.xml
      info.xml
    values/
      strings.xml

```

Sr.No.	Directory & Resource Type
1	anim/ XML files that define property animations. They are saved in res/anim/ folder and accessed from the R.anim class.
2	color/ XML files that define a state list of colors. They are saved in res/color/ and accessed from the R.color class.
3	drawable/ Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the R.drawable class.
4	layout/ XML files that define a user interface layout. They are

	saved in res/layout/ and accessed from the R.layout class.
5	menu/ XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the R.menu class.
6	raw/ Arbitrary files to save in their raw form. You need to call <i>Resources.openRawResource()</i> with the resource ID, which is <i>R.raw.filename</i> to open such raw files.
7	values/ XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory – <ul style="list-style-type: none">• arrays.xml for resource arrays, and accessed from the R.array class.• integers.xml for resource integers, and accessed from the R.integer class.• bools.xml for resource boolean, and accessed from the R.bool class.• colors.xml for color values, and accessed from the R.color class.• dimens.xml for dimension values, and accessed from the R.dimen class.• strings.xml for string values, and accessed from the R.string class.• styles.xml for styles, and accessed from the R.style class.

xml/

Arbitrary XML files that can be read at runtime by calling `Resources.getXML()`. You can save various configuration files here which will be used at run time.

Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e.images) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow the following steps –

- Create a new directory in res/ named in the form `<resources_name>-<config_qualifier>`.

Here **resources_name** will be any of the resources mentioned in the above table, like layout, drawable etc. The **qualifier** will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.

- Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Below is an example which specifies images for a default screen and alternative images for high resolution screen.

MyProject/

app/

```
manifest/  
    AndroidManifest.xml  
java/  
    MyActivity.java  
res/  
    drawable/  
        icon.png  
        background.png  
drawable-hdpi/  
        icon.png  
        background.png  
    layout/  
        activity_main.xml  
        info.xml  
    values/  
        strings.xml
```

Below is another example which specifies layout for a default language and alternative layout for Arabic language.

```
MyProject/  
    app/  
        manifest/  
            AndroidManifest.xml  
    java/  
        MyActivity.java  
res/  
    drawable/  
        icon.png  
        background.png  
drawable-hdpi/  
        icon.png  
        background.png  
    layout/  
        activity_main.xml  
        info.xml  
layout-ar/
```

```
main.xml  
values/  
strings.xml
```

Accessing Resources

During your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios –

Accessing Resources in Code

When your Android application is compiled, a **R** class gets generated, which contains resource IDs for all the resources available in your **res/** directory. You can use R class to access that resource using sub-directory and resource name or directly resource ID.

Example

To access *res/drawable/myimage.png* and set an ImageView you will use following code –

```
ImageView imageView = (ImageView)  
findViewById(R.id.myimageview);  
imageView.setImageResource(R.drawable.myimage);
```

Here first line of the code make use of *R.id.myimageview* to get ImageView defined with id *myimageview* in a Layout file. Second line of code makes use of *R.drawable.myimage* to get an image with name **myimage** available in drawable sub-directory under **/res**.

Example

Consider next example where *res/values/strings.xml* has following definition –

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="hello">Hello, World!</string>  
</resources>
```

Now you can set the text on a TextView object with ID msg using a resource ID as follows –

```
TextView msgTextView = (TextView) findViewById(R.id.msg);  
msgTextView.setText(R.string.hello);
```

Example

Consider a layout *res/layout/activity_main.xml* with the following definition –

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:orientation="vertical" >  
  
<TextView android:id="@+id/text"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello, I am a TextView" />  
  
<Button android:id="@+id/button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello, I am a Button" />  
  
</LinearLayout>
```

This application code will load this layout for an Activity, in the `onCreate()` method as follows –

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

Consider the following resource XML *res/values/strings.xml* file that includes a color resource and a string resource –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="opaque_red">#f00</color>
    <string name="hello">Hello!</string>
</resources>
```

Now you can use these resources in the following layout file to set the text color and text string as follows –

```
<?xml version="1.0" encoding="utf-8"?>
<EditText
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

Now if you will go through previous chapter once again where I have explained **Hello World!** example, and I'm sure you will have better understanding on all the concepts explained in this chapter. So I highly recommend to check previous chapter for working example and check how I have used various resources at very basic level.

CHAPTER 5

EVENT DRIVEN CONTROLS

5.1 SWITCHING STATES WITH TOGGLE BUTTONS

A ToggleButton displays checked/unchecked states as a button. It is basically an on/off button with a light indicator.



ToggleButton

ToggleButton Attributes

Following are the important attributes related to ToggleButton control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes at run time.

Sr.No.	Attribute & Description
1	android:disabledAlpha This is the alpha to apply to the indicator when disabled.
2	android:textOff This is the text for the button when it is not checked.
3	android:textOn This is the text for the button when it is checked.

Inherited from android.widget.TextView Class –

Sr.No.	Attribute & Description
1	android:autoText If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	android:drawableBottom This is the drawable to be drawn below the text.
3	android:drawableRight This is the drawable to be drawn to the right of the text.
4	android:editable If set, specifies that this TextView has an input method.
5	android:text This is the Text to display.

Inherited from android.view.View Class –

Sr.No.	Attribute & Description
1	android:background This is a drawable to use as the background.
2	android:contentDescription This defines text that briefly describes content of the view.
3	android:id This supplies an identifier name for this view,
4	android:onClick This is the name of the method in this View's context to invoke when the view is clicked.
5	android:visibility This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and ToggleButton.

- | Step | Description |
|------|---|
| 1 | You will use Android studio IDE to create an Android application and name it as My Application under a package com.example.saira_000.myapplication as explained in the Hello World Example chapter. |
| 2 | Modify src/MainActivity.java file to add a click event. |
| 3 | Modify the default content of res/layout/activity_main.xml file to include Android UI control. |
| 4 | No need to declare default constants.Android studio takes care of default constants at string.xml |
| 5 | Run the application to launch Android emulator and verify the result of the |

changes done in the application.

Following is the content of the modified main activity file **src/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.saira_000.myapplication;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends ActionBarActivity {
    ToggleButton tg1,tg2;
    Button b1;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tg1=(ToggleButton)findViewById(R.id.toggleButton);
        tg2=(ToggleButton)findViewById(R.id.toggleButton2);

        b1=(Button)findViewById(R.id.button2);
        b1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                StringBuffer result = new StringBuffer();
                result.append("You have clicked first ON Button-:");
                ").append(tg1.getText());
                result.append("You have clicked Second ON Button -:");
                ").append(tg2.getText());
                Toast.makeText(MainActivity.this,
                result.toString(),Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Following will be the content of res/layout/activity_main.xml file –

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials point"
        android:textColor="#ff87ff09"
        android:textSize="30dp"
        android:layout_above="@+id/imageButton"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="40dp" />

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageButton"
        android:src="@drawable/abc"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

    <ToggleButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="On"
        android:id="@+id/toggleButton"
        android:checked="true"
        android:layout_below="@+id/imageButton"
        android:layout_toEndOf="@+id/button2"/>

    <ToggleButton
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Off"
        android:id="@+id/toggleButton2"
        android:checked="true"
        android:layout_alignTop="@+id/toggleButton" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:text="ClickMe"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />

</RelativeLayout>

```

Following will be the content of res/values/strings.xml to define these new constants –

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
</resources>

```

Following is the default content of AndroidManifest.xml –

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.saira_000.myapplication" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.My Application.MainActivity"
            android:label="@string/app_name" >

```

```

<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>

</activity>
</application>
</manifest>

```

Let's try to run your My Application application. I assume you had created your AVD while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window

—

The following screen will appear



If you have clicked first on Button, you would get a message on Toast as You have clicked first ON Button-) or else if you clicked on second on button, you would get a message on Toast as You have clicked Second ON Button -:)

Android ToggleButton Example

Android Toggle Button can be used to display checked/unchecked (On/Off) state on the button.

It is beneficial if user have to change the setting between two states. It can be used to On/Off Sound, Wifi, Bluetooth etc.

Since Android 4.0, there is another type of toggle button called switch that provides slider control.

Android ToggleButton and Switch both are the subclasses of CompoundButton class.

Android ToggleButton class

ToggleButton class provides the facility of creating the toggle button.

XML Attributes of ToggleButton class

The 3 XML attributes of ToggleButton class.

XML Attribute	Description
android:disabledAlpha	The alpha to apply to the indicator when disabled.
android:textOff	The text for the button when it is not checked.
android:textOn	The text for the button when it is checked.

Methods of ToggleButton class

The widely used methods of ToggleButton class are given below.

Method	Description
CharSequence getTextOff()	Returns the text when button is not in the checked state.
CharSequence getTextOn()	Returns the text for when button is in the checked state.
void setChecked(boolean checked)	Changes the checked state of this button.

Android ToggleButton Example

activity_main.xml

Drag two toggle button and one button for the layout. Now the activity_main.xml file will look like this:

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="example.javatpoint.com.togglebutton.MainActivity">

<ToggleButton
    android:id="@+id/toggleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="80dp"
    android:text="ToggleButton"
    android:textOff="Off"
    android:textOn="On"
    app:layout_constraintEnd_toStartOf="@+id/toggleButton2"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<ToggleButton
    android:id="@+id/toggleButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginRight="60dp"
    android:layout_marginTop="80dp"
    android:text="ToggleButton"
    android:textOff="Off"
    android:textOn="On"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="144dp"
    android:layout_marginLeft="148dp"
    android:text="Submit"
    app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Activity class

Let's write the code to check which toggle button is ON/OFF.

File: MainActivity.java

```
package example.javatpoint.com.togglebutton;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {
    private ToggleButton toggleButton1, toggleButton2;
    private Button buttonSubmit;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        addListenerOnButtonClick();
    }

    public void addListenerOnButtonClick(){
        //Getting the ToggleButton and Button instance from the layout xml file
        toggleButton1=(ToggleButton)findViewById(R.id.toggleButton);
        toggleButton2=(ToggleButton)findViewById(R.id.toggleButton2);
        buttonSubmit=(Button)findViewById(R.id.button);

        //Performing action on button click
        buttonSubmit.setOnClickListener(new View.OnClickListener(){

            @Override
            public void onClick(View view) {
```

```

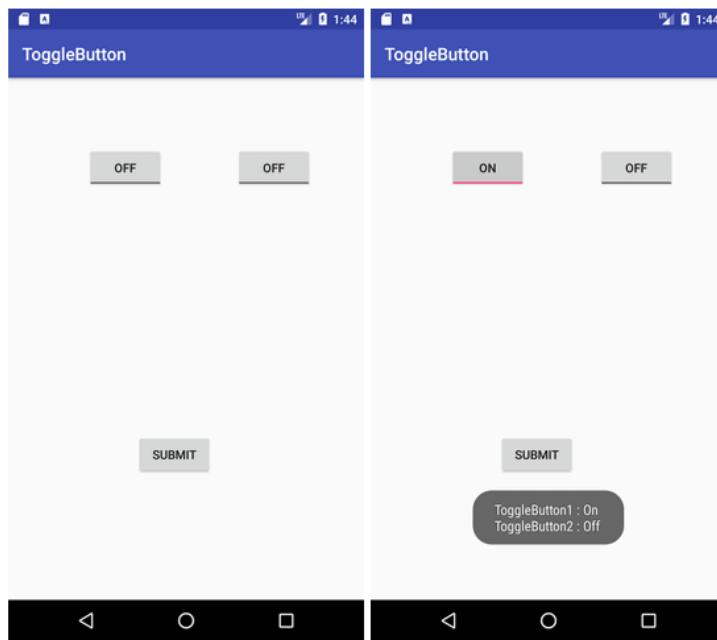
        StringBuider result = new StringBuider();
        result.append("ToggleButton1 : ").append(toggleButton1.getText());
        result.append("\nToggleButton2 : ").append(toggleButton2.getText());
        //Displaying the message in toast
        Toast.makeText(getApplicationContext(),
result.toString(),Toast.LENGTH_LONG).show();
    }

});
```

}

}

Output:



5.2 CREATING AN IMAGES SWITCHER APPLICATION

Sometimes you don't want an image to appear abruptly on the screen; rather you want to apply some kind of animation to the image when it transitions from one image to another. This is supported by android in the form of ImageSwitcher.

An image switcher allows you to add some transitions on the images through the way they appear on screen. In order to use image Switcher, you need to define its XML component first. Its syntax is given below –

```
<ImageSwitcher  
    android:id="@+id/imageSwitcher1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:layout_centerVertical="true" >  
</ImageSwitcher>
```

Now we create an instance of ImageSwitcher in java file and get a reference of this XML component. Its syntax is given below –

```
private ImageSwitcher imageSwitcher;  
imageSwitcher = (ImageSwitcher) findViewById(R.id.imageSwitcher1);
```

The next thing we need to do implement the ViewFactory interface and implement unimplemented method that returns an imageView. Its syntax is below –

```
imageSwitcher.setImageResource(R.drawable.ic_launcher);  
imageSwitcher.setFactory(new ViewFactory() {  
    public View makeView() {  
        ImageView myView = new ImageView(getApplicationContext());  
        return myView;  
    }  
})
```

The last thing you need to do is to add Animation to the ImageSwitcher. You need to define an object of Animation class through AnimationUtilities class by calling a static method loadAnimation. Its syntax is given below –

```
Animation in =  
AnimationUtils.loadAnimation(this, android.R.anim.slide_in_left);  
imageSwitcher.setInAnimation(in);  
imageSwitcher.setOutAnimation(out);
```

The method setInAnimation sets the animation of the appearance of the object on the screen whereas setOutAnimation does the opposite. The method loadAnimation() creates an animation object.

Apart from these methods, there are other methods defined in the ImageSwitcher class. They are defined below –

Sr.No	Method & description
1 setImageDrawable(Drawable drawable)	Sets an image with image switcher. The image is passed in the form of bitmap
2 setImageResource(int resid)	Sets an image with image switcher. The image is passed in the form of integer id
3 setImageURI(Uri uri)	Sets an image with image switcher. The image is passed in the form of URI
4 ImageSwitcher(Context context, AttributeSet attrs)	Returns an image switcher object with already setting some attributes passed in the method
5 onInitializeAccessibilityEvent (AccessibilityEvent event)	Initializes an AccessibilityEvent with information about this View which is the event source
6 onInitializeAccessibilityNodeInfo (AccessibilityNodeInfo info)	Initializes an AccessibilityNodeInfo with information about this view

Example

The below example demonstrates some of the image switcher effects on the bitmap. It creates a basic application that allows you to view the animation effects on the images.

To experiment with this example , you need to run this on an actual device.

Steps	Description
1	You will use Android studio IDE to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components
4	Run the application and choose a running android device and install the application on it and verify the results

Following is the content of the modified main activity file **src/MainActivity.java**.

“In the below code tp and abc indicates the logo “

```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.app.ActionBar.LayoutParams;
import android.os.Bundle;
import android.view.View;

import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.Toast;
import android.widget.ViewSwitcher.ViewFactory;

public class MainActivity extends Activity {
    private ImageSwitcher sw;
    private Button b1,b2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        b1 = (Button) findViewById(R.id.button);
        b2 = (Button) findViewById(R.id.button2);

        sw = (ImageSwitcher) findViewById(R.id.imageSwitcher);
        sw.setFactory(new ViewFactory() {
```

```

@Override
public View makeView() {
    ImageView myView = new ImageView(getApplicationContext());
    myView.setScaleType(ImageView.ScaleType.FIT_CENTER);
    myView.setLayoutParams(new
        ImageSwitcher.LayoutParams(LayoutParams.WRAP_CONTENT,
        LayoutParams.WRAP_CONTENT));
    return myView;
}
});

b1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "previous Image",
        Toast.LENGTH_LONG).show();
        sw.setImageResource(R.drawable.abc);
    }
});

b2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "Next Image",
        Toast.LENGTH_LONG).show();
        sw.setImageResource(R.drawable.tp);
    }
});
}

```

*Following is the modified content of the xml **res/layout/activity_main.xml**.*

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

```

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

<TextView android:text="Gestures Example"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textview"
    android:textSize="35dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point"
    android:id="@+id/textView"
    android:layout_below="@+id/textview"
    android:layout_centerHorizontal="true"
    android:textColor="#ff7aff24"
    android:textSize="35dp" />

<ImageSwitcher
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageSwitcher"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="168dp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/left"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true" />
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/right"  
    android:id="@+id/button2"  
    android:layout_alignParentBottom="true"  
    android:layout_alignLeft="@+id/button"  
    android:layout_alignStart="@+id/button" />  
  
</RelativeLayout>
```

Following is the content of **Strings.xml** file.

```
<resources>  
    <string name="app_name">My Application</string>  
    <string name="left"><![CDATA[<]]></string>  
    <string name="right"><![CDATA[>]]></string>  
</resources>
```

Following is the content of **AndroidManifest.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.example.sairamkrishna.myapplication"  
  
    <application  
        android:allowBackup="true"  
        android:icon="@drawable/ic_launcher"  
        android:label="@string/app_name"  
        android:theme="@style/AppTheme" >  
  
        <activity  
            android:name="com.example.sairamkrishna.myapplication.MainActivity"  
            android:label="@string/app_name" >  
  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>
```

```
</activity>
```

```
</application>
```

```
</manifest>
```

Let's try to run your application we just modified. I assume you had created your AVD while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window

5.3 SCROLLING THROUGH SCROLL VIEW

In android, ScrollView is a kind of layout that is useful to add vertical or horizontal scroll bars to the content which is larger than the actual size of layouts such as linearlayout, relativelayout, framelayou, etc.

Generally, the android ScrollView is useful when we have content that doesn't fit our android app layout screen. The ScrollView will enable a scroll to the content which is exceeding the screen layout and allow users to see the complete content by scrolling.

The android ScrollView can hold only one direct child. In case, if we want to add multiple views within the scroll view, then we need to include them in another standard layout like linearlayout, relativelayout, framelayou, etc.

To enable scrolling for our android applications, ScrollView is the best option but we should not use ScrollView along with ListView or Gridview because they both will take care of their own vertical scrolling.

In android, ScrollView supports only vertical scrolling. In case, if we want to implement horizontal scrolling, then we need to use a HorizontalScrollView component.

The android ScrollView is having a property called android:fillViewport, which is used to define whether the ScrollView should stretch its content to fill the viewport or not.

Now we will see how to use ScrollView with linearlayout to enable scroll view to the content which is larger than screen

Android ScrollView Example

Following is the example of enabling vertical scrolling to the content which is larger than the layout screen using an android ScrollView object.

Create a new android application using android studio and give names as ScrollViewExample. In case if you are not aware of creating an app in android studio check this article [Android Hello World App](#).

Once we create an application, open activity_main.xml file from \res\layout folder path and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fillViewport="false">
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TextView android:id="@+id/loginscrn"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="80dp"
            android:text="ScrollView"
            android:textSize="25dp"
            android:textStyle="bold"
            android:layout_gravity="center"/>
        <TextView android:id="@+id/fstTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:text="Welcome to Tutlane"
            android:layout_gravity="center"/>
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginTop="60dp"
            android:text="Button One" />
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginTop="60dp"
```

```
        android:text="Button Two" />
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="60dp"
    android:text="Button Three" />
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="60dp"
    android:text="Button Four" />
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="60dp"
    android:text="Button Five" />
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="60dp"
    android:text="Button Six" />
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="60dp"
    android:text="Button Seven" />
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="60dp"
    android:text="Button Eight" />
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="60dp"
    android:text="Button Nine" />
</LinearLayout>
</ScrollView>
```

If you observe above code, we used a ScrollView to enable the scrolling for

linearLayout whenever the content exceeds layout screen.

Android HorizontalScrollView Example

Now open activity_main.xml file in your android application and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<HorizontalScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fillViewport="true">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="150dp">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button One" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Two" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Three" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Four" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Five" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Six" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Seven" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

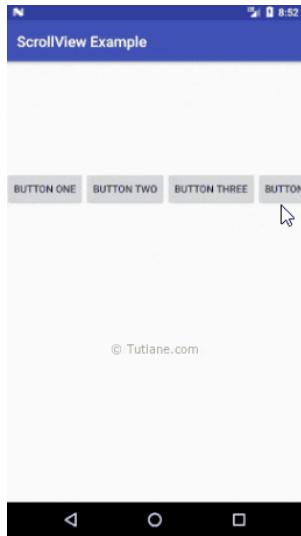
```

        android:text="Button Eight" />
<Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Nine" />
</LinearLayout>
</HorizontalScrollView>
```

If you observe above code, we used a HorizontalScrollView to enable horizontal scrolling for linearlayout whenever the content exceeds layout screen

Output of Android HorizontalScrollView Example

When we run the above example in the android emulator we will get a result like as shown below.



If you observe above result, HorizontalScrollView provided a horizontal scrolling for linearlayout whenever the content exceeds the layout screen.

This is how we can enable scrolling for the content which exceeds layout screen using ScrollView and HorizontalScrollView object based on our requirements.

5.4 PLAYING AUDIO AND VIDEO

Android provides many ways to control playback of audio/video files and streams. One of this way is through a class called MediaPlayer.

Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio,video e.t.c. In order to use MediaPlayer, we have to call a static Method create() of this class. This method returns an instance of MediaPlayer class. Its syntax is as follows –

```
MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);
```

The second parameter is the name of the song that you want to play. You have to make a new folder under your project with name raw and place the music file into it.

Once you have created the Mediaplayer object you can call some methods to start or stop the music. These methods are listed below.

```
mediaPlayer.start();  
mediaPlayer.pause();
```

On call to start() method, the music will start playing from the beginning. If this method is called again after the pause() method, the music would start playing from where it is left and not from the beginning.

In order to start music from the beginning, you have to call reset() method. Its syntax is given below.

```
mediaPlayer.reset();
```

Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below –

Sr.No	Method & description
1	isPlaying() This method just returns true/false indicating the song is playing or not
2	seekTo(position) This method takes an integer, and move song to that particular position millisecond
3	getCurrentPosition() This method returns the current position of song in milliseconds
4	getDuration() This method returns the total time duration of song in milliseconds
5	reset()

	This method resets the media player	
6	release() This method releases any resource attached with MediaPlayer object	
7	setVolume(float leftVolume, float rightVolume) This method sets the up down volume for this player	
8	setDataSource(FileDescriptor fd) This method sets the data source of audio/video file	
9	selectTrack(int index) This method takes an integer, and select the track from the list on that particular index	
10	getTrackInfo() This method returns an array of track information	

Example

Here is an example demonstrating the use of MediaPlayer class. It creates a basic media player that allows you to forward, backward, play and pause a song.

To experiment with this example, you need to run this on an actual device to hear the audio sound.

Steps	Description	
1	You will use Android studio IDE to create an Android application under a package com.example.sairamkrishna.myapplication.	

2	Modify src/MainActivity.java file to add MediaPlayer code.	
3	Modify the res/layout/activity_main to add respective XML components	
4	Create a new folder under MediaPlayer with name as raw and place an mp3 music file in it with name as song.mp3	
5	Run the application and choose a running android device and install the application on it and verify the results	

Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;
import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;
import java.util.concurrent.TimeUnit;
```

```
public class MainActivity extends Activity {
```

```
    private Button b1,b2,b3,b4;
    private ImageView iv;
    private MediaPlayer mediaPlayer;
```

```
    private double startTime = 0;
    private double finalTime = 0;
```

```
    private Handler myHandler = new Handler();
    private int forwardTime = 5000;
```

```

private int backwardTime = 5000;
private SeekBar seekbar;
private TextView tx1,tx2,tx3;

public static int oneTimeOnly = 0;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b1 = (Button) findViewById(R.id.button);
    b2 = (Button) findViewById(R.id.button2);
    b3 = (Button) findViewById(R.id.button3);
    b4 = (Button) findViewById(R.id.button4);
    iv = (ImageView) findViewById(R.id.imageView);

    tx1 = (TextView) findViewById(R.id.textView2);
    tx2 = (TextView) findViewById(R.id.textView3);
    tx3 = (TextView) findViewById(R.id.textView4);
    tx3.setText("Song.mp3");

    mediaPlayer = MediaPlayer.create(this, R.raw.song);
    seekbar = (SeekBar) findViewById(R.id.seekBar);
    seekbar.setClickable(false);
    b2.setEnabled(false);

    b3.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(getApplicationContext(), "Playing
                sound", Toast.LENGTH_SHORT).show();
            mediaPlayer.start();

            finalTime = mediaPlayer.getDuration();
            startTime = mediaPlayer.getCurrentPosition();

            if (oneTimeOnly == 0) {
                seekbar.setMax((int) finalTime);
                oneTimeOnly = 1;
            }
        }
    });
}

```

```

tx2.setText(String.format("%d min, %d sec",
    TimeUnit.MILLISECONDS.toMinutes((long) finalTime),
    TimeUnit.MILLISECONDS.toSeconds((long) finalTime) -
    TimeUnit.MINUTES.getSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
        finalTime)))
);

tx1.setText(String.format("%d min, %d sec",
    TimeUnit.MILLISECONDS.toMinutes((long) startTime),
    TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
    TimeUnit.MINUTES.getSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
        startTime)))
);

seekbar.setProgress((int)startTime);
myHandler.postDelayed(UpdateSongTime,100);
b2.setEnabled(true);
b3.setEnabled(false);
}
});

b2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    Toast.makeText(getApplicationContext(), "Pausing
        sound",Toast.LENGTH_SHORT).show();
    mediaPlayer.pause();
    b2.setEnabled(false);
    b3.setEnabled(true);
}
});

b1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    int temp = (int)startTime;

```

```

        if((temp+forwardTime)<=finalTime){
            startTime = startTime + forwardTime;
            mediaPlayer.seekTo((int) startTime);
            Toast.makeText(getApplicationContext(),"You have Jumped forward 5
                seconds",Toast.LENGTH_SHORT).show();
        }else{
            Toast.makeText(getApplicationContext(),"Cannot jump forward 5
                seconds",Toast.LENGTH_SHORT).show();
        }
    });
}

b4.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int temp = (int)startTime;

        if((temp-backwardTime)>0){
            startTime = startTime - backwardTime;
            mediaPlayer.seekTo((int) startTime);
            Toast.makeText(getApplicationContext(),"You have Jumped backward 5
                seconds",Toast.LENGTH_SHORT).show();
        }else{
            Toast.makeText(getApplicationContext(),"Cannot jump backward 5
                seconds",Toast.LENGTH_SHORT).show();
        }
    });
}

private Runnable UpdateSongTime = new Runnable() {
    public void run() {
        startTime = mediaPlayer.getCurrentPosition();
        tx1.setText(String.format("%d min, %d sec",
            TimeUnit.MILLISECONDS.toMinutes((long) startTime),
            TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
            TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.
            toMinutes((long) startTime)))
    );
        seekbar.setProgress((int)startTime);
    }
}

```

```

        myHandler.postDelayed(this, 100);
    }
};

}

```

Following is the modified content of the xml **res/layout/activity_main.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView android:text="Music Player" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textview"
        android:textSize="35dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials point"
        android:id="@+id/textView"
        android:layout_below="@+id/textview"
        android:layout_centerHorizontal="true"
        android:textColor="#ff7aff24"
        android:textSize="35dp" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"

```

```
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:src="@drawable/abc"/> 
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/forward"
    android:id="@+id/button"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pause"
    android:id="@+id/button2"
    android:layout_alignParentBottom="true"
    android:layout_alignLeft="@+id/imageView"
    android:layout_alignStart="@+id/imageView" />
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/back"
    android:id="@+id/button3"
    android:layout_alignTop="@+id/button2"
    android:layout_toRightOf="@+id/button2"
    android:layout_toEndOf="@+id/button2" />
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/rewind"
    android:id="@+id/button4"
    android:layout_alignTop="@+id/button3"
    android:layout_toRightOf="@+id/button3"
    android:layout_toEndOf="@+id/button3" />
```

```
<SeekBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/seekBar"
    android:layout_alignLeft="@+id/textview"
    android:layout_alignStart="@+id/textview"
    android:layout_alignRight="@+id/textview"
    android:layout_alignEnd="@+id/textview"
    android:layout_above="@+id/button" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView2"
    android:layout_above="@+id/seekBar"
    android:layout_toLeftOf="@+id/textView"
    android:layout_toStartOf="@+id/textView" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView3"
    android:layout_above="@+id/seekBar"
    android:layout_alignRight="@+id/button4"
    android:layout_alignEnd="@+id/button4" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Medium Text"
    android:id="@+id/textView4"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_centerHorizontal="true" />
```

```
</RelativeLayout>
```

Following is the content of the **res/values/string.xml**

```
<resources>
    <string name="app_name">My Application</string>
    <string name="back"><![CDATA[<]]></string>
    <string name="rewind"><![CDATA[<<]]></string>
    <string name="forward"><![CDATA[>>]]></string>
    <string name="pause">||</string>
</resources>
```

Following is the content of **AndroidManifest.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sairamkrishna.myapplication" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.sairamkrishna.myapplication.MainActivity"
            android:label="@string/app_name" >

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>
    </application>
</manifest>
```

Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your

project's activity files and click Run Eclipse Run Icon from the toolbar. Before starting your application, Android studio will display following screens

5.5 ALL DISPLAYING PROGRESS WITH PROGRESS BAR

We can display the **android progress bar** dialog box to display the status of work being done e.g. downloading file, analyzing status of work etc.

In this example, we are displaying the progress dialog for dummy file download operation.

Here we are using **android.appProgressDialog** class to show the progress bar. Android ProgressDialog is the subclass of AlertDialog class.

Android Progress Bar Example by ProgressDialog

Let's see a simple example to create progress bar using ProgressDialog class.

activity_main.xml

Drag one button from the pallete, now the activity_main.xml file will look like this:

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/andro  
id"
```

```
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity" >
```

<Button

```
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentTop="true"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="116dp"
```

```
        android:text="download file" />  
  
</RelativeLayout>
```

Activity class

Let's write the code to display the progress bar dialog box.

File: MainActivity.java

```
import android.app.ProgressDialog;  
import android.os.Handler;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
  
public class MainActivity extends AppCompatActivity {  
    Button btnStartProgress;  
    ProgressDialog progressBar;  
    private int progressBarStatus = 0;  
    private Handler progressBarHandler = new Handler();  
    private long fileSize = 0;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        addListenerOnButtonClick();  
    }  
    public void addListenerOnButtonClick() {  
        btnStartProgress = findViewById(R.id.button);  
        btnStartProgress.setOnClickListener(new View.OnClickListener(){  
  
            @Override  
            public void onClick(View v) {
```

```

// creating progress bar dialog
progressBar = new ProgressDialog(v.getContext());
progressBar.setCancelable(true);
progressBar.setMessage("File downloading ...");
progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);

progressBar.setProgress(0);
progressBar.setMax(100);
progressBar.show();
//reset progress bar and filesize status
progressBarStatus = 0;
fileSize = 0;

new Thread(new Runnable() {
    public void run() {
        while (progressBarStatus < 100) {
            // performing operation
            progressBarStatus = doOperation();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            // Updating the progress bar
            progressBarHandler.post(new Runnable() {
                public void run() {
                    progressBar.setProgress(progressBarStatus);
                }
            });
        }
        // performing operation if file is downloaded,
        if (progressBarStatus >= 100) {
            // sleeping for 1 second after operation completed
            try {

```

```
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    // close the progress bar dialog
    progressBar.dismiss();
}
}).start();
}//end of onClick method
});
}
```

CHAPTER 6

MENUS

In almost all applications we encounter menus that display options in the form of menu items. Choosing a menu item results in the initiation of the desired task. Menus are therefore the preferred way of indicating the possible actions in an application.

The ActionBar is a widget that replaces the title bar at the top of an Activity displaying navigation and important functionality of an application. It provides a consistent UI of an application. It also displays the key actions that are commonly used in an application and that are constantly visible on the screen. ActionBar is also commonly used to provide a quick link to an application's home.

6.1 MENUS AND THEIR TYPES

Android SDK supports three types of menus: Options, Submenu, and Context, as detailed in the following list:

- **Options Menu**—Also known as the Activity menu, this menu is displayed when a MENU button is clicked. In an Options Menu, the menu items are displayed in the form of text, check box, or radio buttons. It can display shortcut keys but does not display icons. In older Android API levels (Levels 10 and below), there were two types of Options Menus:
- **Icon Menu**—The Icon Menu appears when the user presses the MENU button on the device. The Icon Menu shows the first six menu items of the menu in the form of large, finger-friendly buttons arranged in a grid at the bottom of the screen. The menu items in the Icon Menu can display icons as well as text. The Icon Menu does not display check boxes, radio buttons, or the shortcut keys for menu items. It is not mandatory to have icons for Icon Menu items; text will also do for the Icon Menu.
- **Expanded Menu**—If the menu has more than six menu items, the first five items are displayed as an Icon Menu and the sixth option appears as a More button. Clicking the More button displays the Expanded Menu that shows the scrollable list of the rest of the menu items that could not be accommodated in the Icon Menu. The Expanded Menu can display menu items in the form of text, check boxes, or radio buttons. Also, it can display shortcut keys but does not display icons. Pressing the Back button from the Expanded Menu takes you back to the Activity the menu was launched from.
- **Submenu**—Submenu refers to the menu that displays more detailed or specific menu options when a menu item is selected. A submenu is displayed as a floating window showing all of its menu options. The name of the submenu is shown in the header bar, and each menu option is displayed with its full text, check box, radio button (if any), and shortcut key. Icons are not displayed in the submenu options. We cannot add a submenu to any menu option of a submenu as Android does not support nested

submenus. Pressing the Back button closes the floating window without navigating back to the Expanded or Icon menus.

- **Context Menu**—The Context Menu is displayed when we tap-and-hold on the concerned View or when a user holds the middle D-pad button or presses the trackball. Context Menus support submenus, check boxes, radio buttons, and shortcuts, but we cannot display icons in a Context Menu. In the Context Menu's header bar, we can display a title and icon.

6.2 CREATING MENUS THROUGH XML

Besides through Java code, in Android, menus can be created through Resources too. We can define a menu in the form of an XML file, which is then loaded by the Android SDK. As usual, Android generates resource IDs for each of the loaded menu items. The XML file for the menu has to be kept in a specific, designated folder that does not exist by default; we need to create it manually.

To understand the procedure of creating menus practically, let's create a new Android application called MenuApp. We see that a folder called menu already exists in our res/ directory. The menu folder also contains a file activity_menu_app.xml by default. We learn to create all three types of menus: Options Menu, Submenu, and Context Menu. Let's begin with the Options Menu.

Creating an Options Menu

An Options Menu is the menu displayed when the MENU button on the device is clicked. It shows menu items in the form of text, check boxes, and radio buttons. Also, the shortcut keys can be assigned to the menu items. An Options Menu is defined through an XML file in the res/menu folder of the application. That is, the menu items of the Options Menu are defined in that XML file.

The activity_menu_app.xml file, which is automatically created for us, contains the following data:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/menu_settings" android:title="@string/menu_settings"
      android:orderInCategory="100" android:showAsAction="never" />
</menu>
```

We can see that the root node of our menu in the XML file is `<menu>`. The default menu item with the ID `menu_settings` shows the text that is defined in the `strings.xml` file. Let's define certain `<item>` nodes to display certain menu items. The code written in the `activity_menu_app.xml` file is as shown in Listing.

Listing . The Menu Items Defined in the File activity_menu_app.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/create_datab" android:title="Create Database"
    android:icon="@drawable/ic_launcher" />
<item android:id="@+id/insert_rows" android:title="Insert Rows"
    android:icon="@drawable/ic_launcher" />
<item android:id="@+id/list_rows" android:title="List Rows" />
<item android:id="@+id/search_row" android:title="Search"
    android:icon="@drawable/ic_launcher"/>
<item android:id="@+id/delete_row" android:title="Delete" />
<item android:id="@+id/update_row" android:title="Update"
    android:icon="@drawable/ic_launcher" />
</menu>
```

We can see that we have defined six menu items in the preceding menu, and a unique ID is assigned to each of them. The android:title and android:icon attribute defines the text and icon for the menu item. The menu items defined in our menu file are Create Database, Insert

Rows, List Rows, Search, Delete, and Update. The six menu items are assigned the IDs as create_datab, insert_rows, list_rows, search_row, delete_row, and update_row, respectively. In the application, we want to display a message that asks the user to select the MENU button on the device or emulator for displaying the Icon Menu. We use the TextView control to display messages. To define TextView, we write the code as shown in Listing in the layout file activity_menu_app.xml.

Listing. The Layout File activity_menu_app.xml on Adding the TextView Control

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/selectedopt" />
</LinearLayout>
```

We added a TextView with the ID selectedopt to the default layout LinearLayout that we use to display desired text messages to the user. The TextView also is used to inform which menu item is selected by the user. To inflate or merge the menu that we defined in the mymenu.xml file in our application and also to inform which menu item is selected by the user, we write the Java code as shown in Listing in our Activity file MenuAppActivity.java.

Listing. Code Written in the Java Activity File MenuAppActivity.java

```
package com.androidunleashed.menuapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MenuInflater;
import android.widget.TextView;

public class MenuAppActivity extends Activity { private TextView selectedOpt;

@Override
public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
setContentView(R.layout.activity_menu_app);
selectedOpt=(TextView)findViewById(R.id.selectedopt); selectedOpt.setText("Please
select MENU button to display menu");
}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{ MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.activity_menu_app,
menu); return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item)
{ switch (item.getItemId()) {
case R.id.create_datab:
selectedOpt.setText("You have selected Create Database option"); break;
case R.id.insert_rows:
selectedOpt.setText("You have selected Insert Rows option"); break;
case R.id.list_rows:
selectedOpt.setText("You have selected List Rows option"); break;
case R.id.search_row:
selectedOpt.setText("You have selected Search Row option"); break;
case R.id.delete_row:
selectedOpt.setText("You have selected Delete Row option"); break;
case R.id.update_row:
selectedOpt.setText("You have selected Update Row option"); break;
}
}
}
```

```

selectedOpt.setText("You have selected Update Row option"); break;
}
return true;
}
}

```

We capture the TextView defined in the layout and assign it to a TextView object named selectedOpt. We use this TextView for displaying an initial message and for informing which menu item is selected by the user. The onCreateOptionsMenu() method of the Activity is called when the user clicks the MENU button of the device. So, we override this method to display our Icon Menu. To display our Icon Menu, we need to inflate or merge our menu that we defined in the activity_menu_app.xml file in the menu provided as a parameter to this method. Initially, there is no menu item defined in the menu parameter. To inflate the menu that we defined in the activity_menu_app.xml file, we get the MenuInflater from the Activity class. An object, inflater, is created of the MenuInflater class and the inflater's inflate method is called to inflate, or merge, our own menu defined in the activity_menu_app.xml file to the menu given as a parameter to this method. The onCreateOptionsMenu() method is set to return the Boolean value true to allow Android to display the menu. By now, we have learned to display an Options Menu consisting of six menu items. But the menu is of no use if no action takes place on selecting its menu items. We have not yet assigned any task to the menu items of our Options Menu. Hence, the next step for us is to define the action to take place when the user selects any of the menu items.

Handling Menu Selections

All the menu items selected in the Activity's menu are handled through the onOptionsItemSelected() method. The selected menu item is passed to this method as the MenuItem parameter. We override this method in the Activity to write the code that we want to execute when a menu item is selected. In the method, we extract the Menu Item ID of the menu item selected for identifying it and then can take the respective action. The getItemId() method helps in knowing the ID of the selected menu item.

The following code is part of the previous application and explains how the action is taken when a menu item is selected:

```

public boolean onOptionsItemSelected(MenuItem item)
{
    switch (item.getItemId())
    {
        case R.id.create_datab:
            selectedOpt.setText("You have selected Create Database
option"); break;
    }
}

```

```

.....
.....
.....
}

return true;
}

```

The preceding method is automatically invoked when a menu item is selected from the menu. The selected menu item is passed to this method and assigned to the MenuItem object item. In preceding code example, getItemId() method is called to know the ID of the selected menu item, and through the switch statement respective action is taken on the selected menu item. In this code, we are setting a text message through the TextView object selectedOpt to inform the user which of the menu items is selected.

On running the application, we get a TextView showing the message Please select MENU button to display menu (see Figure —left). When we select the MENU button of the device or emulator, the Options Menu is displayed showing the respective menu items. When we select a menu item, say Create Database, the TextView responds by showing the message You have selected Create Database option, as shown in Figure (right).



Figure. A TextView directing the user to select the MENU button to invoke a menu (left), and the TextView displays the option selected from the Options menu (right).

Similarly, when we select the Delete menu item, the TextView responds by showing the message. You have selected Delete Row option, as shown in Figure.

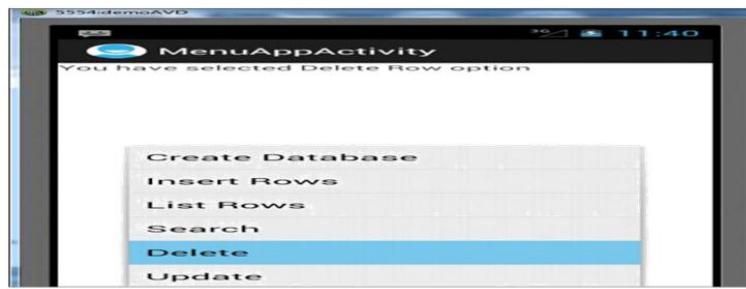


Figure. The message displayed through the TextView control changes on selecting another option from the Icon menu.

By now, we know how to display simple text as menu items and also how the menu selections are handled. We can also define check boxes, shortcuts, and radio buttons in menu items. Let's learn how.

Defining Check Boxes and Shortcuts

In this section, we learn how to define checkable menu items and shortcuts for them. We add more menu items to the menu in our application MenuApp. One of the newly added menu items will be checkable, and the other will be invoked with assigned shortcut keys. The menu file activity_menu_app.xml is modified to appear as shown in Listing. Only the code in bold is newly added; the rest of the code is the same as first Listing

Listing. The Menu File activity_menu_app.xml After Defining Checkable Menu Items and Shortcuts

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/create_database"
        android:title="Create Database"
        android:icon="@drawable/ic_launcher" />
    <item android:id="@+id/insert_rows"
        android:title="Insert Rows"
        android:icon="@drawable/ic_launcher" />
    <item
        android:id="@+id/list_rows"
        android:title="List Rows"
        />
    <item android:id="@+id/search_row"
        android:title="Search"
        android:icon="@drawable/ic_launcher"/>
    <item
        android:id="@+id/delete_row"
        android:title="Delete" />
```

```

<item android:id="@+id/update_row"
    android:title="Update"
    android:icon="@drawable/ic_launcher" />
<item
    android:id="@+id/sort_rows"
    android:title="Sort Table"
    android:checkable="true"
    android:checked="true"/>
<item
    android:id="@+id/merge_row"
    android:title="Merge Rows"
    android:alphabeticShortcut="m"
    android:numericShortcut="4"/>
</menu>

```

We can see that the code now has eight menu items. The seventh menu item, sort_rows, is set to appear in the form of a check box via the android:checkable attribute. When we set the value of the android:checkable property to true, the menu item appears as a check box. The check box can be initially set to appear in either a checked or unchecked state by the android:checked attribute. When we set android:checked="true", the menu item appears as checked by default.

Shortcut keys are assigned to the last menu item, Merge Rows (merge_row). When we set the android:alphabeticShortcut="m" attribute, the shortcut key m for the full keyboard is assigned to the menu item. Similarly, the android:numericShortcut="4" attribute assigns the shortcut key 4 from the numeric keypad. The shortcut keys are displayed below the menu item text. When the menu is open (or while holding the MENU key), if we press the m key from the full keyboard or 4 from the numeric keypad, the menu item Merge Rows is selected.

On running the application, we get a TextView showing the message Please select MENU button to display menu. When the MENU button of the device or emulator is selected, the Options Menu is displayed, showing respective menu items (see Figure). We can see that the menu item Sort Table appears as a checkable menu item. When we select any menu item, the respective text message is displayed on the screen. For example, if Sort Table is selected, the TextView responds by showing the message You have selected Sort Table option. If we press the m or 4 key while holding the MENU key, the menu item Merge Rows is selected (as shortcut keys are assigned to this menu item).



Figure. The Options Menu showing the text and checkable menu items

Note

Android API Level 10 and below categorize the Options Menu as an Icon Menu and an Expanded menu. If the menu items are less than or equal to six, the menu is called an Icon Menu; otherwise it is an Expanded Menu. An Icon Menu displays menu items in the form of icons and text, whereas the Expanded Menu displays menu items in the form of a scrollable list and shows only the menu items that could not fit in the Icon Menu. An Expanded Menu automatically appears when more than six menu items are defined in the Options Menu. When an Icon Menu with more than six menu items is displayed, a More button appears as a sixth menu item. When we select the More button, the rest of the menu items appear in the Expanded Menu. In the Expanded Menu, when we click the Back button we return to the Activity the Icon Menu was launched from. Expanded Menu items support check boxes, radio buttons, or shortcut keys; Icon Menu items do not.

While creating menu items, we may need a menu item with submenu items. Let's see how submenus are added.

Adding Submenus

A submenu is a collection of menu items invoked when a regular menu item is selected. Usually, a submenu represents more detailed options for the selected menu item. The submenu can be associated with any menu item. A submenu is created by enclosing `<item>` nodes within the

`<menu>` node, where `<item>` nodes represent the submenu options and the `<menu>` node acts as the root node of the submenu. To associate a submenu with any menu item, just put the

`<menu>` node of the submenu along with its nested `<item>` nodes inside the `<item>` node of the menu item to which we want to assign the submenu.

Let's add a submenu consisting of three menu items—Search on Code, Search on Name, and Search on Price—to the Search menu. To add a submenu to the Search menu item, `activity_menu_app.xml` is modified to appear as shown in Listing. The code in bold is newly added; the rest is the same as previous Listing.

Listing. The Code in the Menu File activity_menu_app.xml After Adding a Submenu

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/create_datab"
    android:title="Create Database"
    android:icon="@drawable/ic_launcher" />
<item android:id="@+id/insert_rows"
    android:title="Insert Rows"
    android:icon="@drawable/ic_launcher" />
<item
    android:id="@+id/list_rows"
    android:title="List Rows" />
<item android:id="@+id/search_row"
    android:title="Search"
    android:icon="@drawable/ic_launcher">
    <menu>
        <group android:checkableBehavior="single">
            <item android:id="@+id/search_code"
                android:title="Search on Code"
                android:checked="true" />
            <item android:id="@+id/search_name"
                android:title="Search on Name"
                android:alphabeticShortcut="n"
                android:numericShortcut="6" />
            <item android:id="@+id/search_price"
                android:title="Search on Price" />
        </group>
    </menu>
</item>
<item
    android:id="@+id/delete_row"
    android:title="Delete" />
<item android:id="@+id/update_row"
    android:title="Update"
    android:icon="@drawable/ic_launcher" />
<item
    android:id="@+id/sort_rows"
    android:title="Sort Table"
    android:checkable="true"
    android:checked="true" />
```

```

<item
    android:id="@+id/merge_row"
    android:title="Merge Rows"
    android:alphabeticShortcut="m"
    android:numericShortcut="4" />
</menu>

```

We can see that a submenu has been created, consisting of three menu items, Search on Code, Search on Name, and Search on Price, with the IDs search_code, search_name, and search_price, respectively. The submenu is assigned to the Search menu item by nesting its `<menu>` node within the `<item>` node of the Search menu item. If we want the menu items to appear as radio buttons, we need to nest them within the `<group>` node, as we have done with the menu items of our submenu.

The `<group>` node is used to collect certain nodes and for collectively applying attributes to all the nested nodes. The `android:checkableBehavior` attribute determines whether to make the menu items nested within the `<group>` node appear as radio buttons, check boxes, or as simple menu items. The attribute can take three values:

- **single**—Only one item can be checked at a time, producing radio buttons.
- **all**—Any item can be checked, producing check boxes.
- **none**—The item appears as simple text, without a check box or radio button.

We have defined an Expanded Menu and a Submenu for the Search menu item of our Icon Menu with the menu file `mymenu.xml`. Next, we need to write some Java code to add actions to the menu items defined in the submenu. So, in the `MenuAppActivity.java` Java Activity file, we add statements to the `onOptionsItemSelected()` method. These statements display messages showing which menu item from the Expanded Menu or Submenu has been selected. The Activity file appears as shown in Listing. Only the code in bold is newly added; the rest is the same as previous Listing.

Listing. Code Written into the Java Activity File `MenuAppActivity.java`

```

package com.androidunleashed.menuapp; import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MenuInflater; import android.widget.TextView;

public class MenuAppActivity extends Activity { private TextView selectedOpt;

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.main);

```

```

selectedOpt=(TextView)findViewById(R.id.selectedopt); selectedOpt.setText("Please
select MENU button to display menu");
}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{ MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.mymenu, menu);
return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item)
{ switch (item.getItemId()) {
    case R.id.create_datab:
        selectedOpt.setText("You have selected Create Database option"); break;
    case R.id.insert_rows:
        selectedOpt.setText("You have selected Insert Rows option"); break;
    case R.id.list_rows:
        selectedOpt.setText("You have selected List Rows option"); break;
    case R.id.search_row:
        selectedOpt.setText("You have selected Search Row option"); break;
    case R.id.delete_row:
        selectedOpt.setText("You have selected Delete Row option"); break;
    case R.id.update_row:
        selectedOpt.setText("You have selected Update Row option"); break;
    case R.id.sort_rows:
        selectedOpt.setText("You have selected Sort Table option");
        item.setChecked(!item.isChecked());
        break;

    case R.id.merge_row:
        selectedOpt.setText("You have selected Merge Rows option"); break;
    case R.id.search_code:
        selectedOpt.setText("You have selected Search on Code option"); break;
    case R.id.search_name:
        selectedOpt.setText("You have selected Search on Name option"); break;
    case R.id.search_price:
        selectedOpt.setText("You have selected Search on Price option"); break;
    }
return true;
}

```

```
}
```

When the application is run, we see a TextView prompting us to select a MENU button, after which the Icon Menu is displayed. After we select the Search menu item from the Icon Menu, the Searchsubmenu is displayed (see Figure —left). We can select a menu item from the submenu by either clicking it or using its shortcut key (if any). For example, if we select Search on Code as shown in Figure (middle), the TextView responds showing the message, You have selected Search on Code option, as shown in Figure (right).

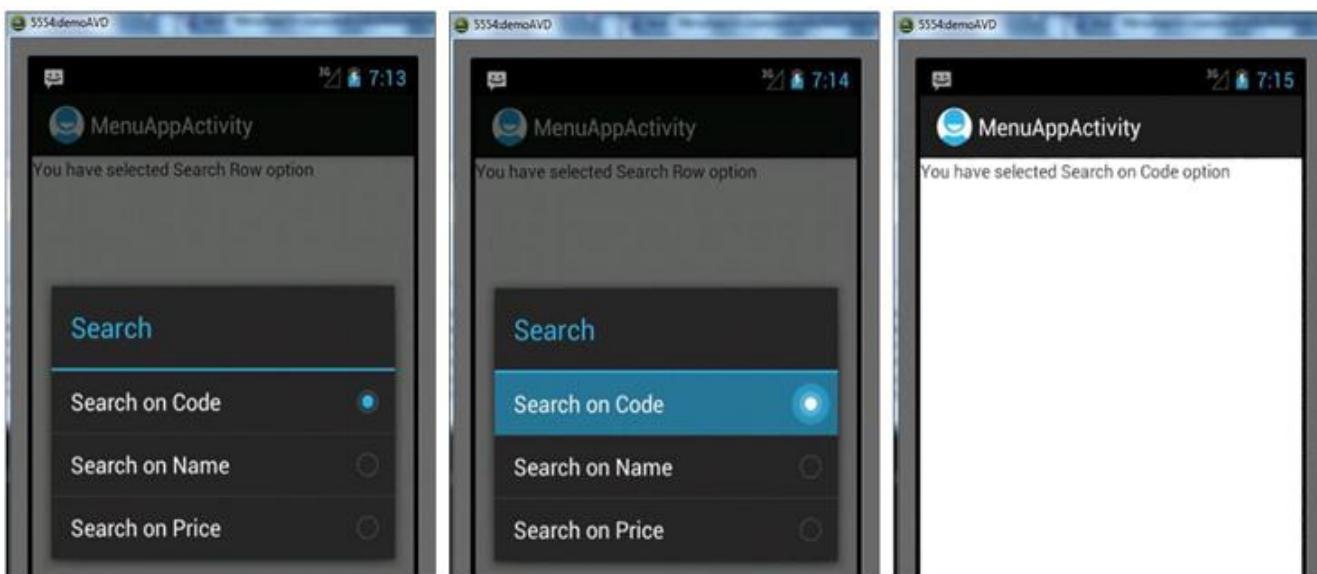


Figure. A submenu appears after selecting a Search menu item from the Icon Menu (left), selecting a menu item from the submenu (middle), and the text message displayed via TextView informs the menu item that was selected (right).

So, we created both an Icon and Expanded Menu and also defined a submenu. Let's create the last menu type, the Context Menu.

Creating a Context Menu

A context menu appears as a floating window and is displayed when the user taps and holds on a widget, holds the middle D-pad button, or presses the trackball. The difference between an options menu and a context menu is that the options menu is invoked when the MENU button is pressed. A context menu appears when we press and hold a View. An Activity can have multiple views, and each view can have its own context menu. An Activity can have only a single options menu but many context menus. Context menus are removed from memory when closed.

Let's define two context menus that we associate with two different TextView controls. For each context menu, we need to add a separate file to the menu folder of our project. Right-click

on the res/menu folder and add two XML files called mycontext_menu1.xml and mycontext_menu2.xml. Let's define three menu items—Cut, Copy, and Find—in the first context menu. The code in mycontext_menu1.xml is shown in Listing

Listing. Code Written into the Context Menu mycontext_menu1.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="all">
        <item android:id="@+id/cut_item"
            android:title="Cut" />
        <item android:id="@+id/copy_item" android:title="Copy" />
    </group>
    <item android:id="@+id/find_item" android:title="Find">
        <menu>
            <item android:id="@+id/find_next" android:title="Find Next" />
        </menu>
    </item>
    </menu>
```

The file shows that three menu items are added to the context menu. The first two are set to appear as check boxes by nesting them within the `<group>` node and applying the `android:checkableBehavior` attribute to them. We just learned that applying the value all to the `android:checkableBehavior` attribute makes the menu items nested in the `<group>` node appear as check boxes. A submenu consisting of a single menu item, Find Next (Find), is attached to the third menu item.

Let's define two menu items in the second context menu. The menu file of the second context menu, mycontext_menu2.xml, appears as shown in Listing.

Listing. Code Written into the Context Menu mycontext_menu2.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/open_item" android:title="Open"
        android:alphabeticShortcut="o" android:numericShortcut="5" />
    <item android:id="@+id/close_item" android:title="Close" android:checkable="true" />
</menu>
```

Listing. The Layout File activity_menu_app.xml After Adding Two TextView Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent" android:layout_height="match_parent" >
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/selectedopt" />
    <TextView
        android:text="View to invoke first context menu"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:id="@+id/contxt1_view" />
    <TextView
        android:text="View to invoke second context menu"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:id="@+id/contxt2_view"/>
</LinearLayout>
```

We can see that two TextView controls were initialized to View to invoke first context menu and View to invoke second context menu. The IDs assigned to these TextView controls are contxt1_view and contxt2_view, respectively.

Next, we need to write some Java code into the Activity file MenuAppActivity.java. To create context menus, we need to override the onCreateOptionsMenu method for registering the Viewsthat are supposed to use it. The option to register a context menu to View(s) enables us to associate context menus for certain selected Views. A context menu is registered to a View through the registerForContextMenu() method (in the Activity file MenuAppActivity.java), as shown in the following code:

```
@Override
public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    selectedOpt=(TextView)findViewById(R.id.selectedopt); selectedOpt.setText("Please
    select MENU button to display menu"); TextView
    contxt1View=(TextView)findViewById(R.id.contxt1_view); TextView
    contxt2View=(TextView)findViewById(R.id.contxt2_view);
    registerForContextMenu(contxt1View); registerForContextMenu(contxt2View);
}
```

We can see that the two TextView controls with the contxt1_view and contxt2_view IDs are captured from the layout file activity_menu_app.xml and mapped to the TextView objects contxt1View and contxt2View, respectively. The two TextView objects are passed to the registerForContextMenu() method, as we want to associate

them with the two context menu just defined.

Once a View is registered, the `onCreateContextMenu()` method is invoked whenever we tap and hold on the registered View. The method also receives a `Menu` object as a parameter that we use to add menu items to it. To add menu items to a menu, the `add()` method is used. We need to override the `onCreateContextMenu()` method for displaying the context menu. The method receives the `menuobject` as a parameter to which we add the menu items for the context menu. Besides the menu object, the method also receives the `View` object that initiated the context menu and a `ContextMenuInfo` object. `ContextMenuInfo` is an interface that belongs to `ContextMenu` and acts as an adapter for passing any other information about menu inflation. The following code snippet is part of the application that creates the context menu. The `onCreateContextMenu()` method appears as shown here:

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
        ContextMenu.ContextMenu- Info menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
    if (v.getId()==R.id.contxt1_view) {  
        MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.mycontext_menu1,  
                menu); menu.setHeaderTitle("Sample Context Menu1");  
        menu.setHeaderIcon(R.drawable.ic_launcher);  
    }  
    if (v.getId()==R.id.contxt2_view) {  
        MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.mycontext_menu2,  
                menu); menu.setHeaderTitle("Sample Context Menu2");  
        menu.setHeaderIcon(R.drawable.ic_launcher);  
    }  
}
```

We can see that we first identify which View has initiated the context menu, and accordingly we inflate or merge the menu that we defined in `mycontext_menu1.xml` or `mycontext_menu2.xml` in the menu provided as the parameter to the `onCreateContextMenu()`

method. To inflate the menu that we defined in the `my context_menu1.xml` and `mycontext_menu2.xml` files, we get the `MenuInflater` from the `Activity` class. A `MenuInflater` class object, `inflater`, is created and its `inflate` method is invoked to inflate, or merge, our own menu, defined in the respective XML file, with the `menu` parameter of this method.

Recall that the context menu supports check boxes, radio buttons, submenus, and

shortcut keys, but not the icons. Two context menus are being created in the preceding code. One is created if the user taps and holds the contxt1View TextView, and the other is created if the user taps and holds the contxt2View TextView. The View.getId() method is used to identify the View that is tapped and held. The first context menu displays two check boxes and a submenu. The second context menu displays two menu items; one is displayed with a shortcut key, and the other is displayed as a check box. The title and icon are set to appear in the context menu's header bar via setHeaderTitle and setHeaderIcon, respectively.

Handling Context Menu Selections

The preferred approach to handling menu items selected in context menus is to override the `onContextItemSelected()` method in the Activity that is automatically invoked whenever a Context Menu Item is selected. The menu item that is selected is passed to the method as a parameter, as shown in the following code:

```
@Override  
public boolean onContextItemSelected(MenuItem item)  
{ switch (item.getItemId()) { case R.id.cut_item:  
selectedOpt.setText("You have selected the Cut option");  
item.setChecked(!item.isChecked()); break;  
  
.....  
.....  
}  
return true;  
}
```

The Menu Item ID of the selected menu item is extracted through the `getItemId()` method, and accordingly the message is displayed through the TextView to display which context menu item is selected. We toggle the state of the checkable item through the `MenuItem.isChecked()` method. The method returns true if the menu item is checked; otherwise it returns false. We change the state of the menu item by passing a Boolean value that represents the reverse of the existing state.

The `MenuAppActivity.java` appears as shown in Listing. Only the code in bold is modified; the rest of the code is the same as we saw in Listing.

Listing. Code Written into the Java Activity File `MenuAppActivity.java`

```
package com.androidunleashed.MenuApp;  
import android.app.Activity;
```

```

import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MenuInflater;
import android.widget.TextView;
import android.view.ContextMenu.ContextMenuItemInfo; import android.view.View;
import android.view.ContextMenu;

public class MenuAppActivity extends Activity { private TextView selectedOpt;

@Override
public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
setContentView(R.layout.main);
selectedOpt=(TextView)findViewById(R.id.selectedopt); selectedOpt.setText("Please
select MENU button to display menu");
TextView contxt1View=(TextView)findViewById(R.id.contxt1_view);
TextView contxt2View=(TextView)findViewById(R.id.contxt2_view);
registerForContextMenu(contxt1View); registerForContextMenu(contxt2View);
}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.mymenu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item)
{ switch (item.getItemId()) {
    case R.id.create_database:
        selectedOpt.setText("You have selected Create Database option"); break;
    case R.id.insert_rows:
        selectedOpt.setText("You have selected Insert Rows option"); break;
    case R.id.list_rows:
        selectedOpt.setText("You have selected List Rows option");
        break;
    case R.id.search_row:
        selectedOpt.setText("You have selected Search Row option"); break;
    case R.id.delete_row:
        selectedOpt.setText("You have selected Delete Row option"); break;
}
}

```

```

selectedOpt.setText("You have selected Delete Row option"); break;
case R.id.update_row:
selectedOpt.setText("You have selected Update Row option"); break;
case R.id.sort_rows:
selectedOpt.setText("You have selected Sort Table option");
item.setChecked(!item.isChecked());
break;
case R.id.merge_row:
selectedOpt.setText("You have selected Merge Rows option"); break;
case R.id.search_code:
selectedOpt.setText("You have selected Search on Code option"); break;
case R.id.search_name:
selectedOpt.setText("You have selected Search on Name option"); break;
case R.id.search_price:
selectedOpt.setText("You have selected Search on Price option"); break;
}
return true;
}

```

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItem
menuInfo)
{
super.onCreateContextMenu(menu, v, menuInfo); if (v.getId()==R.id.context1_view) {
MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.mycontext_menu1,
menu); menu.setHeaderTitle("Sample Context Menu1");
menu.setHeaderIcon(R.drawable.ic_launcher);
}
if (v.getId()==R.id.context2_view)
{
MenuInflater inflater = getMenuInflater();
inflater.inflate(R.menu.mycontext_menu2, menu); menu.setHeaderTitle("Sample
Context Menu2"); menu.setHeaderIcon(R.drawable.ic_launcher);
}
}

```

```

@Override
public boolean onContextItemSelected(MenuItem item)
{ switch (item.getItemId()) { case R.id.cut_item:
selectedOpt.setText("You have selected the Cut option");

```

```

item.setChecked(!item.isChecked());
break;
case R.id.copy_item:
selectedOpt.setText("You have selected the Copy option");
item.setChecked(!item.isChecked()); break;
case R.id.find_item:
selectedOpt.setText("You have selected the Find Submenu"); break;
case R.id.find_next:
selectedOpt.setText("You have selected the Find Next option");
break;
case R.id.open_item:
selectedOpt.setText("You have selected the Open option");
break;
case R.id.close_item:
selectedOpt.setText("You have selected the Close option");
item.setChecked(!item.isChecked());
break;
}
return true;
}
}

```

On running the application, we get three TextView controls. The first one is to tell the user to click the MENU button to display the menu. The second and third TextView controls are meant for invoking context menus. When the TextView controls are tapped and held, the respective Context Menu is displayed (see Figure 5.5—left). After we tap and hold the second TextView, a context menu titled Sample Context Menu1 is displayed, as shown in Figure 5.5 (middle). After we select the Findmenu item from the context menu, a submenu, the Find SubMenu, is displayed, as shown in Figure (right).

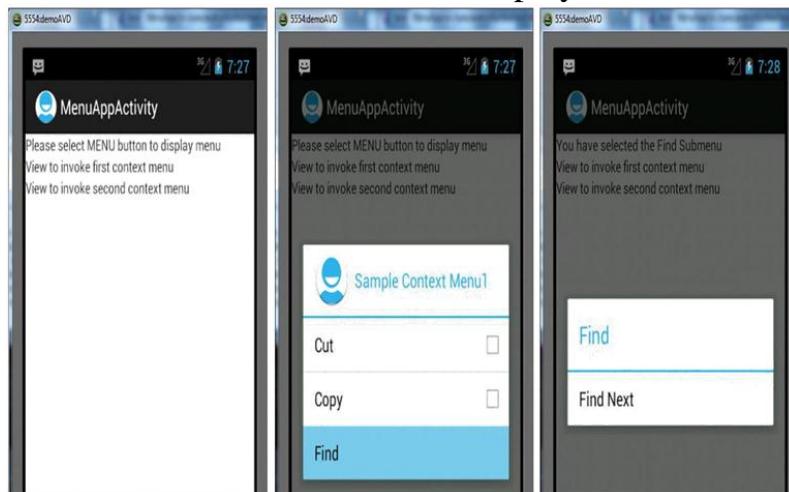


Figure. Three TextView controls showing different messages (left), the first Context Menu containing two check boxes that appear after pressing and holding the second TextView (middle), and a Find submenu appears after selecting a Find menu item (right).

After we tap and hold the third TextView that shows the text View to invoke second context menu (see Figure —left), a context menu, Sample Context Menu2, is displayed, as shown in Figure (middle). After we select the Open item or press its shortcut key, the first TextView responds by displaying the message You have selected the Open option, as shown in Figure (right).

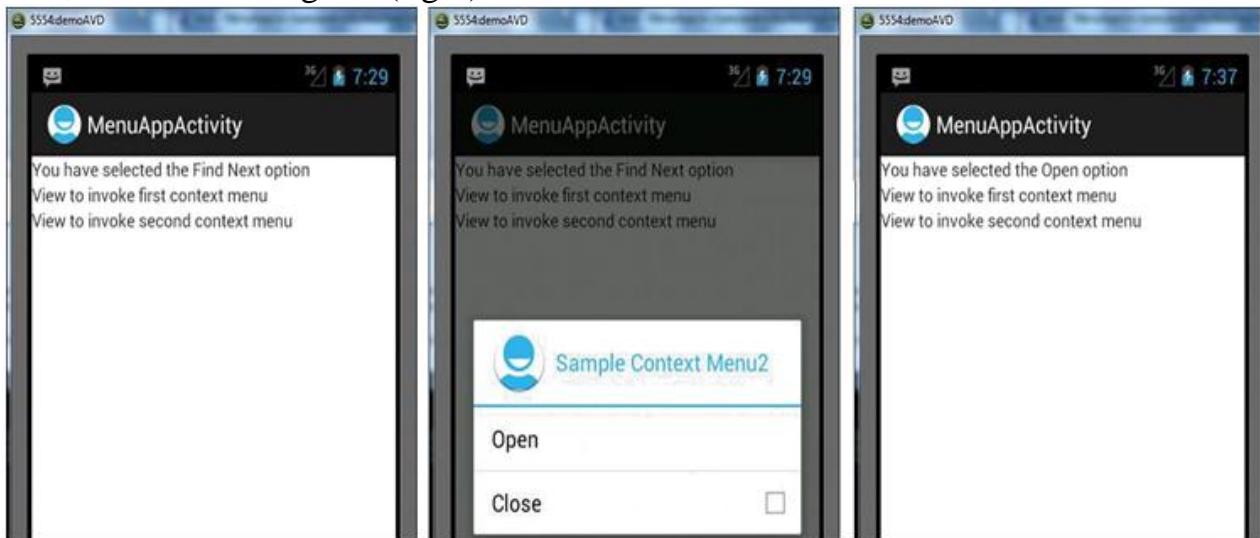


Figure. The first TextView shows that the Find Next option was selected from the Find submenu (left), the second context menu containing a menu item and a check box that appear after pressing and holding the third TextView (middle), and the first TextView shows that the Open menu option was selected from the context menu (right).

Let's create an application that demonstrates the creation of Submenus and context menus through coding instead of by using XML files. But before we do that, let's first look at the different methods that we will encounter.

6.3 CREATING MENUS THROUGH CODING

As stated earlier, besides using XML resources, we can create the menus through Java code too. In this next section, we learn to create the three types of menus, Options Menu, Submenu, and Context Menu, with Java coding. We begin by defining Options Menus.

Defining Options Menus

To define an Options or Activity menu, we override the `onCreateOptionsMenu` method in the Java Activity file `MenuAppCodeActivity.java`. The method receives a menu object as a parameter, which is used to add menu items to it. To add menu items to the menu object, the `add()` method is used.

Here is the syntax:

`add(Group ID, Menu Item ID, Sort order ID, Menu text)`

The `add()` method requires following four parameters:

- **Group ID**—An integer used for grouping a collection of menu items.
- **Menu Item ID**—A unique identifier assigned to each menu item to identify it. Usually a private static variable within the Activity class is assigned to the menu item as a unique identifier. We can also use the `Menu.FIRST` static constant and simply increment that value for successive menu items.
- **Sort order ID**—Defines the order in which the menu items are to be displayed.
- **Menu text**—Text to appear in the menu item.

The following code adds a single menu item, Create Database, to the Options Menu:

```
private static final int CREATE_DATAB = Menu.FIRST;  
menu.add(0,CREATE_DATAB,0,"Create Database").setIcon(R.drawable.ic_launcher);
```

We can see that the `CREATE_DATAB` is assigned a static constant, `Menu.FIRST`, and is used as a Menu Item ID in the `add()` method. In the `add()` method, we can see that Group ID is assigned the value 0. All the menu items that we want to be a part of this group are assigned the Group

ID of 0. The `CREATE_DATAB` is assigned to Menu Item ID; the Sort order ID is 0; and the string `Create Database` is the menu item text. The `setIcon()` method assigns the `ic_launcher.png` image to the menu item from the drawable resource (the `res/drawable` directory).

Note

We can avoid the `setIcon()` method if we want only a menu item text without an icon.

Assigning Icons

We can assign icons to the menu items in the Icon Menu using the `setIcon()` method.

Here is the syntax:

`menuItem.setIcon(R.drawable.icon_filename);`

where `icon_filename` is a drawable resource identifier for the icon to be assigned to the menu item. Icons are only displayed in the Icon Menu and not visible in Extended,

Submenu, and Contextmenus.

After we add all the menu and submenu items to the menu, the `onCreateOptionsMenu()` method is set to return the Boolean value true to allow Android to display the menu.

Creating Submenus

Submenus appear as single floating windows displaying all of their menu items. A submenu is attached to a regular menu item that, when selected, invokes the submenu, hence displaying all the menu items in it. A submenu is added through the `addSubMenu()` method. It supports the same parameters as the `add()` method used to add menu items in the Options Menu: Group ID, Menu Item ID, Sort order ID, and Menu Text. We can also use the `setHeaderIcon` method to specify an icon to display in the submenu's header bar and the `setIcon()` method to display the icon for the menu item to which this submenu is associated.

Note

Android does not support nested submenus.

The following code adds a submenu to a regular menu item, Search:

```
private static final int SEARCH_ROW = Menu.FIRST + 3;
SubMenu searchSub = menu.addSubMenu(0, SEARCH_ROW, 3, "Search");
searchSub.setHeaderIcon(R.drawable.ic_launcher);
searchSub.setIcon(R.drawable.ic_launcher);
searchSub.add(1, SEARCH_CODE, Menu.NONE, "Search on Code");
```

We can see that a submenu called `searchSub` is created for the regular menu item `Search`. The menu item `Search` to which the submenu `searchSub` is associated is assigned a group ID of 0. `SEARCH_ROW`, which is assigned the value of the static constant `Menu.FIRST+3`, is assigned as the Menu Item ID; 3 is assigned as the Sort order ID; and `Search` is assigned as the menu item text. The `ic_launcher.png` file in Resources is set to display in the submenu's header bar. The image `ic_launcher.png` is set to display as an icon in the `Search` menu item, to which the submenu `searchSub` is associated. A menu item, `Search on Code`, is added to the submenu through the `add()` method.

Using Check Boxes/Radio Buttons in Menus

Android supports check boxes and radio buttons in menu items. To set a menu item as a check box, we use the `setCheckable()` method.

Here is the syntax:

setCheckable(boolean);

The menu item appears as a check box when the Boolean value true is passed to the setCheckable() method. The following code makes a menu item appear as a check box:

```
menu.add(3, CLOSE_ITEM, Menu.NONE, "Close").setCheckable(true);
```

By default, the check box is unchecked. To make the check box checked by default, we use the setChecked() method. By passing the Boolean value true to the setChecked() method, we can make the check box appear checked by default.

The following code makes a menu item appear as a checked check box by default:

```
menu.add(3, CLOSE_ITEM, Menu.NONE,
        "Close").setCheckable(true).setChecked(true);
```

Radio buttons are mutually exclusive menu items displayed as circles; only one menu item can be selected in a group at any time. Selecting a radio button menu item unselects any other previously selected menu items in the same group. To create radio buttons, we need to make them part of the same group; hence we assign the same group identifier to all of them; then we call the setGroupCheckable() method.

Here is the syntax:

setGroupCheckable(int GroupID, boolean Checkable, boolean Exclusive)

where the GroupID refers to the group whose menu items we want to appear as radio buttons or check boxes. The second parameter, Checkable, should be set to true to make the check box or radio button appear as checkable. If we pass a false value to the Checkable parameter, then the menu items appear neither as check boxes nor radio buttons. The third parameter, Exclusive, determines whether we want the menu items to be mutually exclusive. If the Exclusive parameter is set to true, it means the menu items are mutually exclusive and only one can be selected at a time. So we set the parameter Exclusive to true if we want the menu items to appear as radio buttons. Passing the value false to the Exclusive parameter makes all the menu items in the group appear as check boxes.

The following code makes all the menu items of a submenu appear as radio buttons:

```
SubMenu searchSub = menu.addSubMenu(0, SEARCH_ROW, 3, "Search");
searchSub.add(1, SEARCH_CODE, Menu.NONE, "Search on
Code").setChecked(true); searchSub.add(1, SEARCH_NAME, Menu.NONE, "Search on
Name"); searchSub.add(1, SEARCH_PRICE, Menu.NONE, "Search on Price");
searchSub.setGroupCheckable(1, true, true);
```

We can see that a submenu called searchSub is created and is associated with a menu item, Search. The searchSub submenu contains three menu items: Search on Code, Search on Name, and Search on Price. All three menu items are assigned the Group ID 1. All the menu items appear as checkable radio buttons, because the Checkable and Exclusive parameters in the setGroupCheckable() method are passed as true. The first menu item in the group, Search on Code, is set to appear as checked by default by passing the value true to the setChecked() method.

We can also assign shortcut keys to the menu items. Let's see how.

Assigning Shortcut Keys

The methods used to assign shortcut keys are setShortcut, setAlphabeticShortcut, and setNumericShortcut. Let's begin with setShortcut method:

- In the setShortcut() method, we can assign two shortcut keys through this method. One of the shortcut keys is used with the numeric keypad and the second with the full keyboard. Neither key is case sensitive.

Example:

```
menu.add(0,MERGE_ROW,7,"Merge Rows").setShortcut('4', 'm');
```

This code assigns shortcut keys for both modes to the menu item Merge Rows. The number 4 is used as the numeric keypad shortcut, and m acts as the shortcut key while using the full keyboard. This shortcut key also is displayed below the menu item text.

- The setAlphabeticShortcut() and setNumericShortcut() methods can be used to define the shortcut keys for the numeric keypad and full keyboard separately, as shown here:

```
menu.add(0,MERGE_ROW,7,"Merge  
Rows").setAlphabeticShortcut('m').setNumericShortcut('4');
```

We have seen all the methods required to define the Options menu, assign icons to the menu items, create submenus, make menu items appear as check boxes and radio buttons, and assign shortcut keys to the menu items. Let's apply all these methods to create a menu with Java code.

Trying It Out

We have seen all the methods required for defining Options Menus, Submenus, and Context Menus. Also, we have seen the methods required in assigning Icons, using check boxes/radio buttons in menus and assigning shortcut keys to the menu items. It's now time to try out all the knowledge that we have gained. Let's do it. First TextView directs the user to click the MENU button to display the menu. The second and third TextView controls are used for displaying Context Menus. When the user taps and

holds on either TextView control, a Context Menu appears on the screen. To display the three TextView controls, modify the layout file activity_menu_app_code.xml to appear as shown in Listing.

Listing. The LayoutFile activity_menu_app_code.xml After Adding Three TextViewControls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/selectedopt" />
    <TextView
        android:text="View to invoke first context menu"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:id="@+id/contxt1_view" />
    <TextView
        android:text="View to invoke second context menu"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:id="@+id/contxt2_view"/>
</LinearLayout>
```

In the code, we can see that the three TextView controls are assigned the IDs selectedopt, contxt1_view, and contxt2_view, respectively. To tell the user which Views are meant for displaying Context Menus, the TextView controls are set to display the text View to invoke first context menu and View to invoke second context menu, respectively.

To display the menu items in the Options menu, Submenu, and Context Menu, the Activity file MenuAppCodeActivity.java is modified as shown in Listing.

Listing. Code Written into the Java Activity File MenuAppCodeActivity.java

```
package com.androidunleashed.menuappcode;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem; import android.widget.TextView; import
android.view.SubMenu;
import android.view.View;
import android.view.ContextMenu;
```

```
public class MenuAppCodeActivity extends Activity {  
    private static final int CREATE_DATAB = Menu.FIRST;  
    private static final int INSERT_ROWS = Menu.FIRST + 1;  
    private static final int LIST_ROWS = Menu.FIRST+2;  
    private static final int SEARCH_ROW = Menu.FIRST + 3;  
    private static final int DELETE_ROW = Menu.FIRST+4;  
    private static final int UPDATE_ROW = Menu.FIRST + 5;  
    private static final int SORT_ROWS = Menu.FIRST+6;  
    private static final int MERGE_ROW = Menu.FIRST + 7;  
    private static final int SEARCH_CODE = Menu.FIRST + 8;  
    private static final int SEARCH_NAME = Menu.FIRST + 9;  
    private static final int SEARCH_PRICE = Menu.FIRST + 10;  
    private static final int CUT_ITEM = Menu.FIRST + 11;  
    private static final int COPY_ITEM = Menu.FIRST + 12;  
    private static final int OPEN_ITEM = Menu.FIRST + 13;  
    private static final int CLOSE_ITEM = Menu.FIRST + 14;  
    private static final int FIND_ITEM = Menu.FIRST + 15;  
    private static final int FIND_NEXT = Menu.FIRST + 16;  
    private TextView selectedOpt;
```

@Override

```
public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);  
setContentView(R.layout.activity_menu_app_code);  
selectedOpt=(TextView)findViewById(R.id.selectedopt); selectedOpt.setText("Please  
select MENU button to display menu"); TextView  
contxt1View=(TextView)findViewById(R.id.contxt1_view); TextView  
contxt2View=(TextView)findViewById(R.id.contxt2_view);  
registerForContextMenu(contxt1View); registerForContextMenu(contxt2View);  
}
```

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {  
    menu.add(0,CREATE_DATAB,0,"Create  
Database").setIcon(R.drawable.ic_launcher);  
    menu.add(0,INSERT_ROWS,1,"Insert Rows").setIcon(R.drawable.ic_launcher);  
    menu.add(0,LIST_ROWS,2,"List Rows");  
    SubMenu searchSub = menu.addSubMenu(0, SEARCH_ROW, 3,  
    "Search"); menu.add(0,DELETE_ROW,4,"Delete");  
    menu.add(0,UPDATE_ROW,5,"Update");
```

```

menu.add(0,SORT_ROWS,6,"Sort Table").setCheckable(true).setChecked(true);
menu.add(0,MERGE_ROW,7,"Merge
Rows").setAlphabeticShortcut('m').setNumeric Shortcut('4');
searchSub.setHeaderIcon(R.drawable.ic_launcher);
searchSub.setIcon(R.drawable.ic_launcher);
searchSub.add(1, SEARCH_CODE, Menu.NONE, "Search on
Code").setChecked(true);
searchSub.add(1, SEARCH_NAME, Menu.NONE, "Search on
Name").setShortcut('6', 'n');
searchSub.add(1, SEARCH_PRICE, Menu.NONE, "Search on Price");
searchSub.setGroupCheckable(1, true, true);
return true;

}

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    switch
        (item.getItemId
        ()) { case
    CREATE_DAT
    AB:
        selectedOpt.setText("You have selected Create Database option");
        break;
    case INSERT_ROWS:
        selectedOpt.setText("You have selected Insert Rows option");
        break;
    case LIST_ROWS:
        selectedOpt.setText("You have selected List Rows option");
        break;
    case SEARCH_ROW:
        selectedOpt.setText("You have selected Search Submenu
        option"); break;
    case DELETE_ROW:
        selectedOpt.setText("You have selected Delete Row option");
        break;
    case UPDATE_ROW:
        selectedOpt.setText("You have selected Update Row option");
        break;
    case SORT_ROWS:

```

```

selectedOpt.setText("You have selected Sort Table option");
item.setChecked(!item.isChecked());
break;
case MERGE_ROW:
    selectedOpt.setText("You have selected Merge Rows option");
    break;
case SEARCH_CODE:
    selectedOpt.setText("You have selected Search on
Code option"); break;
case SEARCH_NAME:
    selectedOpt.setText("You have selected Search on
Name option"); break;
case SEARCH_PRICE:
    selectedOpt.setText("You have selected Search on Price option");
}
return true;
}

```

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v,
menuInfo);
    if (v.getId()==R.id.context1_view) {
        menu.setHeaderTitle("Sample Context Menu1");
        menu.setHeaderIcon(R.drawable.ic_launcher);
        menu.add(2, CUT_ITEM, Menu.NONE, "Cut");
        menu.add(2, COPY_ITEM, Menu.NONE, "Copy");
        menu.setGroupCheckable(2, true, false);
        SubMenu subcont = menu.addSubMenu(2, FIND_ITEM, Menu.NONE,
"Find"); subcont.add(3, FIND_NEXT, Menu.NONE, "Find Next");
    }
    if (v.getId()==R.id.context2_view) {
        menu.setHeaderTitle("Sample Context Menu2");
        menu.setHeaderIcon(R.drawable.ic_launcher);
        menu.add(3, OPEN_ITEM, Menu.NONE, "Open").setShortcut('5', 'o');
        menu.add(3,
CLOSE_ITEM, Menu.NONE, "Close").setCheckable(true);
    }
}

```

```

        }
    }

@Override
public boolean onContextItemSelected(MenuItem item)
{
    switch
    (item.getItemId
    ()) { case
    CUT_ITEM:
        selectedOpt.setText("You have selected the Cut option");
        item.setChecked(!item.isChecked());
        break;

    case COPY_ITEM:
        selectedOpt.setText("You have selected the Copy option");
        item.setChecked(!item.isChecked());
        break;
    case FIND_ITEM:
        selectedOpt.setText("You have selected the Find Submenu");
        break;
    case FIND_NEXT:
        selectedOpt.setText("You have selected the Find Next
        option"); break;
    case OPEN_ITEM:
        selectedOpt.setText("You have selected the Open option");
        break;
    case CLOSE_ITEM:
        selectedOpt.setText("You have selected the Close option");
        item.setChecked(!item.isChecked());
        break;

    }
    return true;
}
}

```

In the `onCreate()` method shown in the preceding code, we can see that the `TextView` with the ID `selectedOpt` is accessed from the layout file and is mapped to the `TextView` object `selectedOpt`. It is set to display the text Please select MENU button to display

menu. The two TextViewcontrols with the IDs contxt1_view and contxt2_view are registered for displaying a Context Menu.

The onCreateOptionsMenu() method defines the menu items for the Options Menu as well as for the Search SubMenu. The menu items defined for the Options Menu are Create Database, Insert Rows, List Rows, Delete, Update, Sort Table, and Merge Rows. The menu items defined for the Search SubMenu are Search on Code, Search on Name, and Search on Price. Shortcut keys are assigned for the Merge Rows menu item and the Search on Name menu item of the Search SubMenu. The Sort Table menu item is set as a checkable menu item. All the menu items of the Search SubMenu are set to appear as radio buttons.

The onOptionsItemSelected() method informs the menu item that is selected by the user. It displays the text message through the TextView selectedOpt to inform which of the menu options is selected by the user.

The onCreateContextMenu() method displays the two Context Menus. When the user taps and holds on any of the TextView controls with the IDs contxt1_view and contxt2_view the method displays the corresponding Context Menu on the screen. The onContextItemSelected() method does the job of informing us which menu item of the Context Menu is selected by the user.

6.4 APPLYING A CONTEXT MENU TO A LISTVIEW

To display the Context Menu for the item selected from the ListView, let's add Context Menu files to the menu folder. Right-click on the res/menu folder in the Package Explorer window and select the New, Android XML File option. Enter the filename as mycontext_menu1 (without the extension .xml). Keeping the Root Element: as menu (default), click the Finish button to create the context file mycontext_menu1.xml. Repeat the procedure to add one more context file called mycontext_menu2.xml to our menu folder. Write the code as shown in Listing_ in the mycontext_menu1.xml file.

Listing. Code Written into the Context Menu mycontext_menu1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/cut_item"
        android:title="Cut" />
    <item android:id="@+id/copy_item" android:title="Copy" />
</menu>
```

The code in the second context menu file, mycontext_menu2.xml, appears as shown in Listing .

Listing. Code Written into the Second Context Menu mycontext_menu2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/open_item"
        android:title="Open" />
    <item
        android:id="@+id/close
        _item"
        android:title="Close" />
</menu>
```

We want to display a TextView and a ListView in our application. The TextView directs the user to take a desired action and tells the user which item from the ListView is selected by the user. The ListView control is used to display different items on the screen, which the user can tap and

hold to invoke the related Context Menu. To display the TextView and ListView in our application, modify activity_context_menu_app.xml to appear as shown in Listing.

Listing . The Layout File activity_context_menu_app.xml After Adding the TextView and ListView Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent" android:layout_height="match_parent" >
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/selectedopt" />
    <ListView android:id="@+id/listvw"
        android:layout_width="match_parent"      android:layout_height="match_parent"
        android:drawSelectorOnTop="false"/>
</LinearLayout>
```

To add an action, for example, to display the related Context Menu and information about the selected item, let's add the code shown in Listing to the Activity file ContextMenuAppActivity.java.

Listing. Code Written into the Java Activity File ContextMenuAppActivity.java

```
package com.androidunleashed.contextmenuapp;

import android.app.Activity;
```

```

import android.os.Bundle;
import android.view.MenuItem;
import android.view.MenuInflater;
import android.widget.TextView;
import android.view.ContextMenu.ContextMenuItemInfo; import android.view.View;
import android.view.ContextMenu;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.ArrayAdapter;

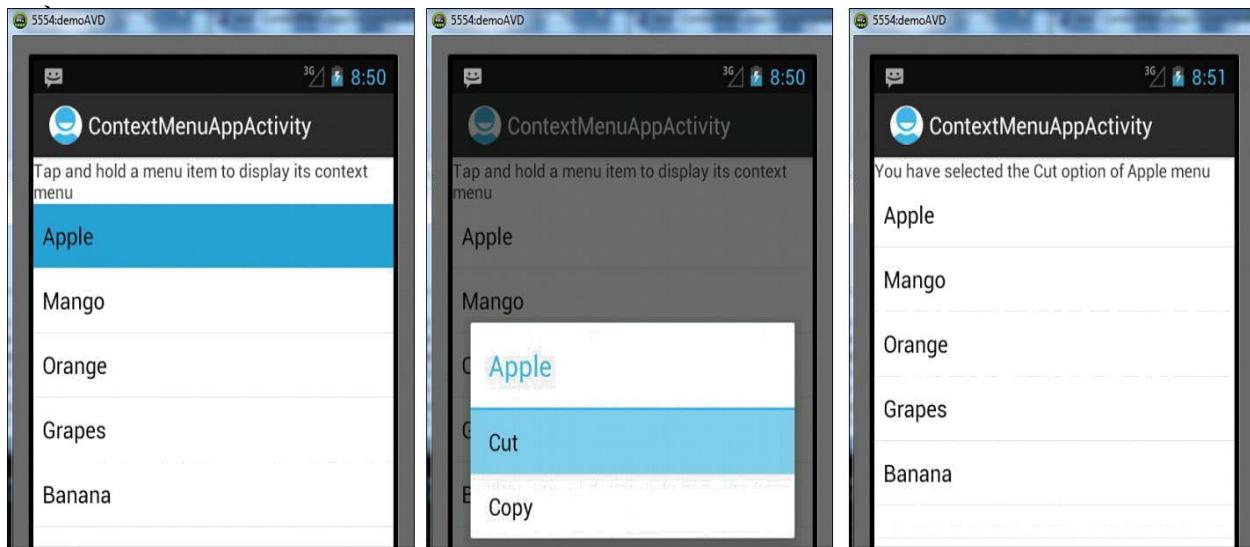
public class ContextMenuAppActivity extends Activity { private TextView
selectedOpt;
String[] fruits={"Apple", "Mango", "Orange", "Grapes", "Banana"};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_context_menu_app);
    selectedOpt=(TextView)findViewById(R.id.selectedopt);
    selectedOpt.setText("Tap and hold a menu item to display its context menu");
    ListView myListview = (ListView)findViewById(R.id.listvw);
    final ArrayAdapter<String> arrayAdpt= new ArrayAdapter<String>(this,
                                            android.R.layout.simple_list_item_1, fruits);
    myListview.setAdapter(arrayAdpt); registerForContextMenu(myListview);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItemInfo
menu- Info)
{
    super.onCreateContextMenu(menu, v, menuInfo); if(v.getId()==R.id.listvw) {
        AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)menuInfo;
        if(fruits[info.position]=="Apple"){ menu.setHeaderTitle(fruits[info.position]);
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.mycontext_menu1, menu);
    }
    if(fruits[info.position]=="Mango") { menu.setHeaderTitle(fruits[info.position]);
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.mycontext_menu2, menu);
    }
}

```

```
}
```



```
        break;
    case R.id.open_item:
        selectedOpt.setText("You have selected the Open option of " +
fruits[info.position]+" menu");
        break;
    case R.id.close_item:
        selectedOpt.setText("You have selected the Close option of the " +
+ fruits[info.position]+" menu"); break;
    }
    return true;
}
}
```

When we run the application, a TextView and a ListView are displayed as shown in Figure (left). When we tap and hold a list item, the related Context Menu appears. For example, when we tap and hold Apple, the Context Menu titled Apple appears with the menu items Cut and Copy, as shown in Figure (middle). When we select a Context Menu Item, a corresponding message is displayed via a TextView. For example, when we select the Cut menu item from the Apple Context Menu, the TextView displays the message You have selected the Cut option of Apple menu, as shown in Figure (right).

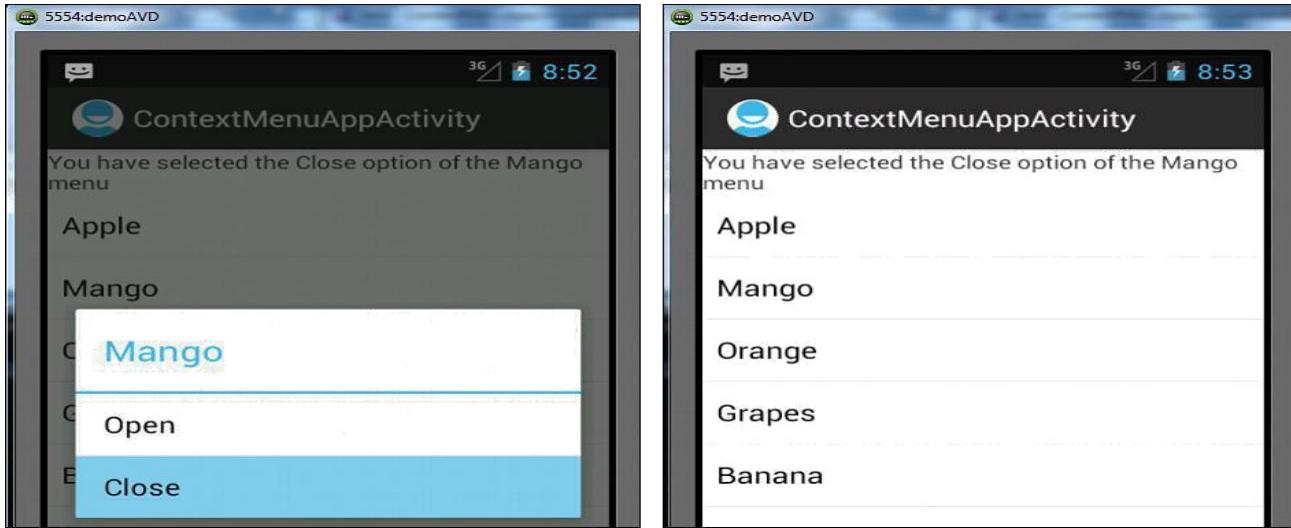


Figure. Items displayed through a ListView and on pressing and holding the Appleitem (left), an Apple Context Menu appears and on selecting the Cut menu item (middle), and the TextView tells the user that the Cut menu item is selected from the Apple context menu (right).

Similarly, when we tap and hold Mango from the ListView, the Context Menu titled Mangoappears with its menu items, Open and Close, as shown in Figure (left).

When we select the Closemenu item from the context menu, the TextView displays the message You have selected the Close option of the Mango menu, as shown in Figure (right).

Figure The Mango Context Menu appears on pressing and holding the Mango item in the ListView (left), and the TextView above the ListView tells the user that the Closemenu item is selected from the Mango Context Menu (right).

We have learned the procedure of creating menus and their role in initiating different tasks. We have also learned the purpose of different menu types including the Options Menu, SubMenu, and Context Menu. The only drawback you might have observed while working with menus is that they are displayed or invoked on pressing the Menu button on the AVD or device. Now, the problem is that many Android devices no longer have a dedicated Menu button. The solution to this problem is using the ActionBar. Making the menu items appear in the ActionBar not only make them instantly

accessible but also removes the need of pressing the Menu button to invoke them. Let's learn more about the ActionBar.

6.5 USING THE ACTIONBAR

The ActionBar is a widget that replaces the title bar at the top of every Activity displaying navigation and important functionality of an application. By default, the ActionBar includes the application logo on the left side, followed by the Activity title, and menu items (if any) on the right side. The ActionBar provides a consistent UI of an application. It helps in displaying the key actions commonly used in an application that we want to be visible on the screen while running the application. The ActionBar is also commonly used to provide a quick link to an application's home. The application's logo acts as a link to the application's home; that is, wherever we are in the application, if we

Note

On Android 3.0 and higher, items from the Options Menu are presented by the ActionBar. It also means that beginning with Android 3.0, the Menu button is deprecated.

tap the application logo displayed through the ActionBar, we navigate to the application's home.

The ActionBar provides the following features:

- Customizes the title bar of an Activity.
- Follows its own life cycle.
- Consistently displays frequently used actions of an application. The menu items from the Options Menu are displayed in the ActionBar to be accessed instantly. The menu items displayed in the ActionBar are also called action items. The menu items that could not be accommodated in the ActionBar appear in the Overflow Menu. The menu items of the Overflow Menu can be seen by selecting the Menu button on the device or the Overflow Menu button in the ActionBar.
- Appears in three forms: standard, tabbed, and list.
- Makes it possible to use the application's icon or logo for navigation.
- Through the ActionBar we can even display Action Views, that is, custom views in the Activity's title bar. For example, we can display the search widget in the ActionBar.

The ActionBar includes the components as shown in [Figure 5.9](#).

- **Application's Icon/Logo**—Displayed at the upper left on the ActionBar.
- **Activity Title**—Displays the title for the ActionBar.
- **Tabs**—Displays the tabs of the ActionBar if the navigation mode set is tabs.
- **Drop-Down List**—Displays the action items in the form of a drop-down list if the navigation mode set is list navigation. We learn about navigation mode soon.
- **ActionItems**—Displays the menu items of the Options menu in the ActionBar.

- **ActionViews**—Displays Custom Views in the ActionBar.
- **Overflow Menu**—Displays menu items that could not be accommodated in the ActionBar.

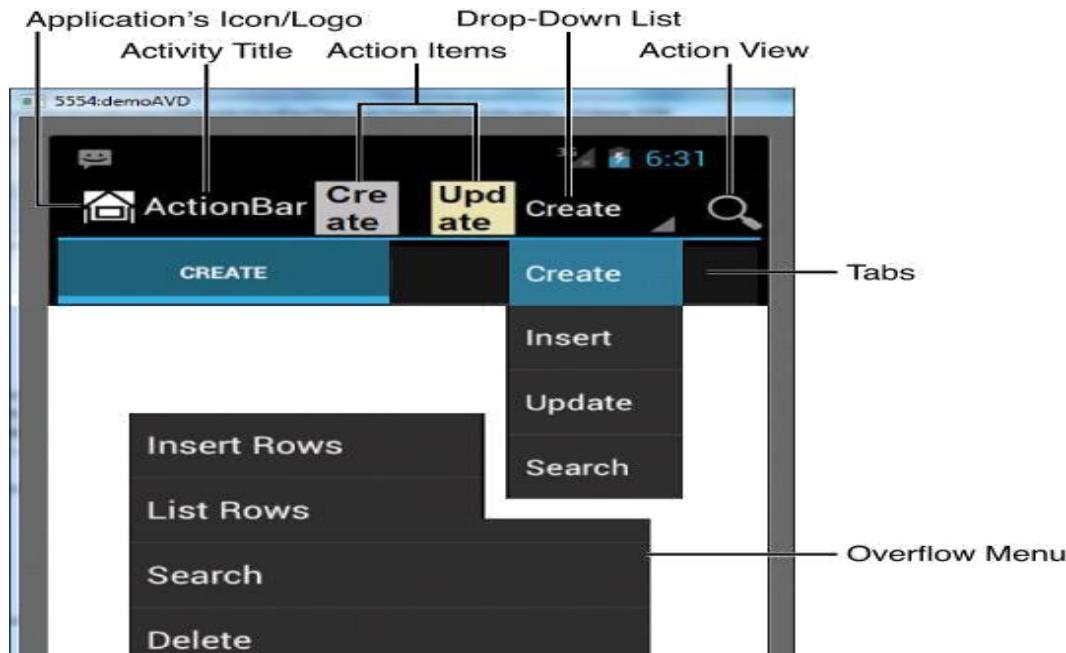


Figure. Different components displayed in anActionBar

Enabling the ActionBar

The ActionBar is enabled if an application uses the default Theme.Holo theme and whose target (or minimum) SDK version is 11 or higher.

Example:

```
<uses-sdk android:targetSdkVersion="15" />
```

To toggle the visibility of an ActionBar at runtime, we can use its show and hide methods as follows:

```
ActionBar actionBar = getActionBar(); actionBar.hide(); // It hides the actionBar
actionBar.show(); // Makes the actionBar visible
```

In the preceding code, the `getActionBar()` method is called to get an `ActionBar` object, and its `hide()` and `show()` methods are for hiding and showing the `ActionBar`, respectively.

To hide the `ActionBar` in an `Activity`, we can also apply a theme that doesn't support it.

In the AndroidManifest.xml file, set the theme for the Activity to Theme.Holo.NoActionBar as shown in the following code:

```
<activity android:label="@string/app_name" android:name=".ActionBarApp"  
        android:theme="@android:style/Theme.Holo.NoActionBar">
```

The icon or logo displayed in the ActionBar can be modified through the android:icon attribute in the configuration file AndroidManifest.xml. We can also use the android:logo attribute in the same file for the same purpose.

The visibility of the icon or logo in the ActionBar is controlled by passing a Boolean value to the setDisplayShowHomeEnabled() method.

The following statement hides the logo or icon in the ActionBar:

```
actionBar.setDisplayShowHomeEnabled(false);
```

The following statement makes the logo or icon visible in the ActionBar:

```
actionBar.setDisplayShowHomeEnabled(true);
```

Similarly, the visibility of the title in the ActionBar can be controlled by passing a Boolean value to the setDisplayShowTitleEnabled() method.

The following statement hides the title in the ActionBar:

```
actionBar.setDisplayShowTitleEnabled(false);
```

The following statement shows the title in the ActionBar:

```
actionBar.setDisplayShowTitleEnabled(true);
```

Using an Application's Icon for Navigation

The logo or icon displayed in an ActionBar if clicked navigates you to the home of the application. “Home of the application” here means the application’s main Activity, that is, the root of our Activity stack.

By default, the logo or icon displayed in the ActionBar is nonclickable. To make the logo or icon clickable, we must call the ActionBar’s setHomeButtonEnabled() method, passing the Boolean value true to it as shown here:

```
actionBar.setHomeButtonEnabled(true);
```

Clicking the logo or icon is considered a Menu Item click. Like Menu Item clicks are handled by the onOptionsItemSelected handler of our Activity; the logo or icon clicks too are handled by the same method. When the logo or icon is clicked, it is considered

that a Menu Item with the ID android.R.id.home is clicked. In other words, when we click the logo or icon, the onOptionItemSelected() method is called passing the Menu Item with the ID android.R.id.home to it as a parameter as shown here:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item)  
{  
    switch (item.getItemId()) {  
        case (android.R.id.home) :  
            Intent intent = new Intent(this, DemoActionBarActivity.class);  
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);  
            startActivity(intent);  
            break;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
    return true;  
}
```

For navigating to the home Activity, we use an intent flag, FLAG_ACTIVITY_CLEAR_TOP, that clears the stack of all activities on top of the home Activity as shown in the preceding code.

Displaying Action Items

To display menu items in the ActionBar as action items, we need to add an android:showAsActionattribute to the menu items while defining them in the menu file. The showAsAction attribute determines how to display the action item. The value of the showAsAction attribute can be any one of the following:

- **always**—Makes the action item appear on the ActionBar.
- **ifRoom**—Makes the action item appear in the ActionBar, but only if there is room available on the ActionBar. If there's not enough room, the item appears in the Overflow Menu.
- **never**—Makes the menu item appear in the Overflow Menu.

To display the Overflow Menu, press the Menu button on the AVD or a soft options menu button that appears as three vertical dots on the physical device.

We need to understand the concept of the ActionBar through a running application. So, let's create a new Android project called DemoActionBar. In this application, we create six Button controls that are used to show/hide the ActionBar, the application's logo, and Activity's title bar. Besides this, the application displays an action item called Create, which when

selected

navigates us to a new Activity, CreateActivity. From the new Activity, when we select the application's logo, we are navigated back to the main Activity. The application also displays an ActionView in the form of a search widget.

After we create the project, the first thing we do is to define six Button controls in the layout file activity_demo_action_bar.xml. These six Button controls are used to show/hide the ActionBar, the application's logo, and the Activity's title bar, respectively. The code written in the layout file activity_demo_action_bar.xml is as shown in Listing.

Listing. Code Written into the Layout File activity_demo_action_bar.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/show_action"
        android:text="Show ActionBar"/>
    <Button android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/hide_action"
        android:text="Hide ActionBar"/>
    <Button android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/show_title"
        android:text="Show Title"/>
    <Button android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/hide_title"
        android:text="Hide Title"/>
    <Button
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:id="@+id/show_logo" android:text="Show Logo"/>
    <Button android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/hide_logo"
        android:text="Hide Logo"/>
</LinearLayout>
```

We can see that the Button controls are assigned the

IDs show_action, hide_action, show_title, hide_title, show_logo, and hide_logo. Also, the caption assigned to the Button controls signifies the task they are supposed to perform. The captions assigned to the Button controls are Show ActionBar, Hide ActionBar, Show Title, Hide Title, Show Logo, and Hide Logo.

Next, we need to define an action item, Create, in our application. This action item

when selected navigates us to the new Activity in the application. The action item is nothing but the menu item from the Options Menu that is displayed in the ActionBar to be accessed instantly. We can define action items in the menu file activity_demo_action_bar.xml that is provided by default in the res/menu folder of our application. In the menu file, we define two menu

items, Create and Search, where Create is displayed in the ActionBar as an action item, and Search is used to display a search widget in the form of ActionView.

The activity_demo_action_bar.xml file, after defining the two menu items in an Options Menu, appears as shown in Listing.

Listing. Code Written into the Menu File activity_demo_action_bar.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/create_datab" android:title="Create"
        android:icon="@drawable/create" android:orderInCategory="0"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_search" android:title="Search"
        android:showAsAction="always"
        android:actionViewClass="android.widget.SearchView"/>
</menu>
```

Because we want to represent the action item Create via an icon, an image file, create.png, should be copied to the res/drawable folders of the application.

On selecting the Create action item, we want to navigate to a new Activity. To define Views for the new Activity, we need to add an XML file to the res/layout folder. So, right-click the layout folder in the Package Explorer window and select the New, Android XML File option. In the dialog box that pops up, specify the filename as create and select the Root Element as LinearLayout as we want the container of this XML file to be LinearLayout. Finally, select the Finish button to create the XML file called create.xml.

When we navigate to the new Activity, we want to display a text message informing that we have navigated to the new Activity. For displaying a text message, we define a TextView control in the new Activity's layout file, create.xml. The layout file create.xml, after defining the TextView control, appears as shown in Listing.

Listing. Code Written into the Layout File create.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content" android:text="This is Create Activity"  
    android:textStyle="bold" />  
</LinearLayout>
```

We can see that the preceding layout file defines a TextView control with the initial text, "This is Create Activity". The text message is displayed when we navigate to the new Activity.

After defining the layout file create.xml for the new Activity, we need a Java class file to load the Views defined in the layout file. So, add a Java class file called CreateActivity.java to the

package com.androidunleashed.demoactionbar of our application. In the CreateActivity.java file, we need to write code to perform the following tasks:

- Load the Views defined in the create.xml file.
 - Make the application's logo clickable. Remember, we want the user to be taken to the main Activity of the application on selecting the application's logo.
 - Navigate to the main Activity file, DemoActionBarActivity.class, of the application.
- For performing all these tasks, we write the code as shown in Listing in the file CreateActivity.java.

Listing. Code Written into the New Activity File CreateActivity.java

```
package com.androidunleashed.demoactionbar;  
  
import android.app.ActionBar;  
import android.app.Activity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.MenuItem;  
  
public class CreateActivity  
  
    extends Activity { @Override  
    protected void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.create);  
        ActionBar actionBar = getActionBar();  
        actionBar.setHomeButtonEnabled(true);  
    }  
  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item)
```

```

    { switch (item.getItemId()) {
        case (android.R.id.home) :
            Intent intent = new Intent(this, DemoActionBarActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
            break;
        default:
            return super.onOptionsItemSelected(item);
    }
    return true;
}
}

```

In the preceding code, we can see that an ActionBar object, actionBar, is accessed by calling the getActionBar() method and then the Boolean value true is passed to the setHomeButtonEnabled() method to make the application's logo clickable. Clicking the application's logo generates a click event on a menu item with the ID android.R.id.home. In the handler method onOptionsItemSelected(), we check whether the menu item with the ID android.R.id.home is clicked, that is, whether the application's logo is clicked. If the application's logo is found to be clicked, we navigate back to the main Activity of the application, DemoActionBarActivity.class, by clearing all the activities (if any) on the top of the stack. For clearing all the top activities, we use an intent flag, FLAG_ACTIVITY_CLEAR_TOP.

We know that no Activity is visible to the Android application until it is mentioned in the configuration file AndroidManifest.xml. To make the newly added Activity CreateActivity.java visible to the Android project, a statement is written in the AndroidManifest.xml file. Also, to replace the default application's logo with a new logo, we need to write code as shown in Listing_ in the AndroidManifest.xml file. Only the statements in bold are added; the rest is the default code.

Listing. Code Written into the Configuration File AndroidManifest.xml

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.DemoActionBar" android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="11" android:targetSdkVersion="15" />
    <application android:icon="@drawable/home"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity android:name=".DemoActionBarActivity"
            android:label="@string/title_activity_demo_action_bar">

```

```

<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=".CreateActivity"
    android:label="@string/app_name"
/></application>
</manifest>

```

Activity, CreateActivity.class, visible to the rest of the application. To enable the ActionBar, the minimum SDK version, that is, the value of the android:minSdkVersion attribute, is set to 11 or higher. Finally, it's time to write code in the main Activity file of the application DemoActionBarActivity.java. We need to perform the following tasks through the main Activity file:

- Access the six Button controls from the layout file activity_demo_action_bar.xml and map them to the respective Button objects.
- Make the three Button controls hide the ActionBar, Activity's title, and application's logo. Also make the hidden components visible through the rest of the three Button controls.
- When the user selects the ActionView, Search widget in the ActionBar, a text box should pop up prompting for the text to search for.
- Navigate to the new Activity CreateActivity.class when the action item Create is selected from the ActionBar.

To perform all these tasks, the code as shown in Listing is written in the main Activity file DemoActionBarActivity.java.

Listing. Code Written into the Java Activity File DemoActionBarActivity.java

```

package com.androidunleashed.demoactionbar; import android.app.Activity;
import android.os.Bundle;
import android.view.View; import android.widget.Button; import
android.app.ActionBar; import android.view.Menu;
import android.view.MenuInflater; import android.view.MenuItem; import
android.content.Intent;

public class DemoActionBarActivity extends Activity
{
    Intent intent;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_demo_action_bar); final ActionBar actionBar

```

```

= getActionBar();
Button showAction = (Button) this.findViewById(R.id.show_action);
Button hideAction = (Button) this.findViewById(R.id.hide_action);
Button showTitle = (Button) this.findViewById(R.id.show_title);
Button hideTitle = (Button) this.findViewById(R.id.hide_title);
Button showLogo = (Button) this.findViewById(R.id.show_logo);
Button hideLogo = (Button) this.findViewById(R.id.hide_logo);

showAction.setOnClickListener(new Button.OnClickListener(){
public void onClick(View v)
{ actionBar.show(); } });
hideAction.setOnClickListener(new Button.OnClickListener(){
public void onClick(View v) {
actionBar.hide(); } });
showTitle.setOnClickListener(new Button.OnClickListener(){
public void onClick(View v)
{ actionBar.setDisplayShowTitleEnabled(true); } });
hideTitle.setOnClickListener(new Button.OnClickListener(){
public void onClick(View v)
{ actionBar.setDisplayShowTitleEnabled(false); } });
showLogo.setOnClickListener(new Button.OnClickListener(){
public void onClick(View v) {
actionBar.setDisplayHomeAsUpEnabled(true); } });
hideLogo.setOnClickListener(new Button.OnClickListener(){
public void onClick(View v)
{ actionBar.setDisplayHomeAsUpEnabled(false); } });

}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{ MenuInflater inflater = getMenuInflater(); inflater.inflate(R.menu.mymenu,
menu);
return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{ switch (item.getItemId()) { case R.id.create_data:
intent = new Intent(this, CreateActivity.class);
startActivity(intent);
break; default:
return super.onOptionsItemSelected(item);
}
}

```

```
    return true;  
}  
}
```

On running the application, we find six Button controls, the action item Create, the ActionView Search widget, and the application's logo on the screen as shown in Figure (left). On selecting the action item Create, we navigate to the Activity CreateActivity. The text message "This is Create Activity" displayed via the TextView defined in the layout file create.xml confirms that we have navigated to the Activity CreateActivity (see Figure — middle). After we select the Button Hide ActionBar, the ActionBar becomes invisible as shown in Figure (right).

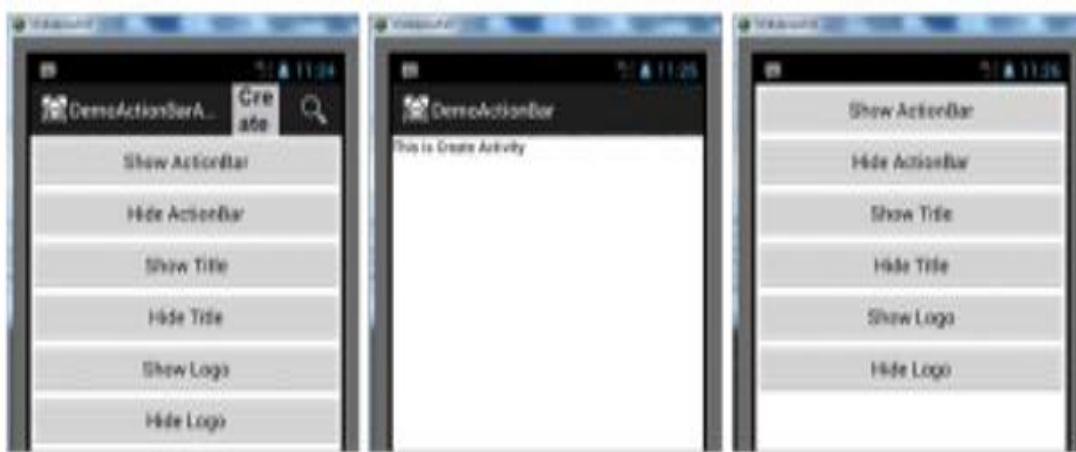


Figure. Screen on startup (left), message displayed in the Create activity (middle), and the screen on hiding the ActionBar (right)

When we select the Button Hide Title, the title of the Activity, DemoActionBar, becomes invisible as shown in Figure 5.11 (left). Similarly, when we select the Button Hide Logo, the application's logo, displayed via home.png file becomes invisible as shown in Figure (middle). When we select the Search widget in the ActionBar, a text box appears prompting us to enter the text we want to search for as shown in Figure (right).



Figure. Activity title is hidden (left), logo is hidden (middle), and a text box opens on selecting the Search widget (right).

6.6 REPLACING A MENU WITH THE ACTIONBAR

We know by now that menus created in the application MenuApp at the beginning of this chapter are useful in initiation of different tasks in an application but need pressing a Menu button for invoking it. If we make the menu items of the menu created in that application appear in the ActionBar, it not only makes the menu items appear consistently throughout the application but also relieves the user from pressing the Menu button. To understand how a menu can be replaced by ActionBar, let's create a new Android project called ActionBarApp. This application displays the simple menu items, checkable menu items, submenu, and so on— everything that we created in the MenuApp application but this time in the ActionBar.

In the menu that we are going to make, we use icon images for a few menu items. So, let's copy and paste four icon images, namely, create.png, insert.png, search.png, and update.png, into the res/drawable folders. The resolution and size of the icons should be as follows:

- **For ldpi**—Resolution should be 120dpi and size $18 \times 18\text{px}$
- **For mdpi**—Resolution should be 160dpi and size $24 \times 24\text{px}$
- **For hdpi**—Resolution should be 240dpi and size $36 \times 36\text{px}$
- **For xhdpi**—Resolution should be 320dpi and size $48 \times 48\text{px}$

Listing. Code Written into the Menu File activity_action_bar_app.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/create_datab" android:title="Create Database"
        android:icon="@drawable/create" android:orderInCategory="0"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/insert_rows" android:title="Insert Rows"
        android:icon="@drawable/insert" android:showAsAction="ifRoom" />
    <item android:id="@+id/list_rows" android:title="List Rows"
        android:showAsAction="ifRoom" />
    <item android:id="@+id/search_row" android:title="Search"
        android:icon="@drawable/search"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/delete_row" android:title="Delete"
        android:showAsAction="never" />
    <item android:id="@+id/update_row" android:title="Update"
        android:icon="@drawable/update" android:showAsAction="always" />
</menu>
```

In the preceding code, we see that the showAsAction attribute is applied to different menu items to determine whether they should appear in the ActionBar or in the Overflow menu. To inflate or merge the menu that we defined in the activity_action_bar_app.xml file in our application, we write the Java code as shown in Listing _ in our Activity file ActionBarAppActivity.java.

Listing. Code Written into the Java Activity File ActionBarAppActivity.java

```
package com.androidunleashed.actionbarapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;

public class ActionBarAppActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_action_bar_app);
    }
}
```

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.activity_action_bar_app, menu);
    return true;
}

```

On running the application, we find that the two menu items, Create and Update, appear in the ActionBar. We can see in the Figure (left) that the icons of the menu items appear without pressing the Menu button. The rest of the menu items that could not be accommodated in the ActionBar are displayed in the Overflow Menu as shown in Figure (right). No text of the action items appears but only the icons in the ActionBar, as there is not enough space when the device is in portrait mode.

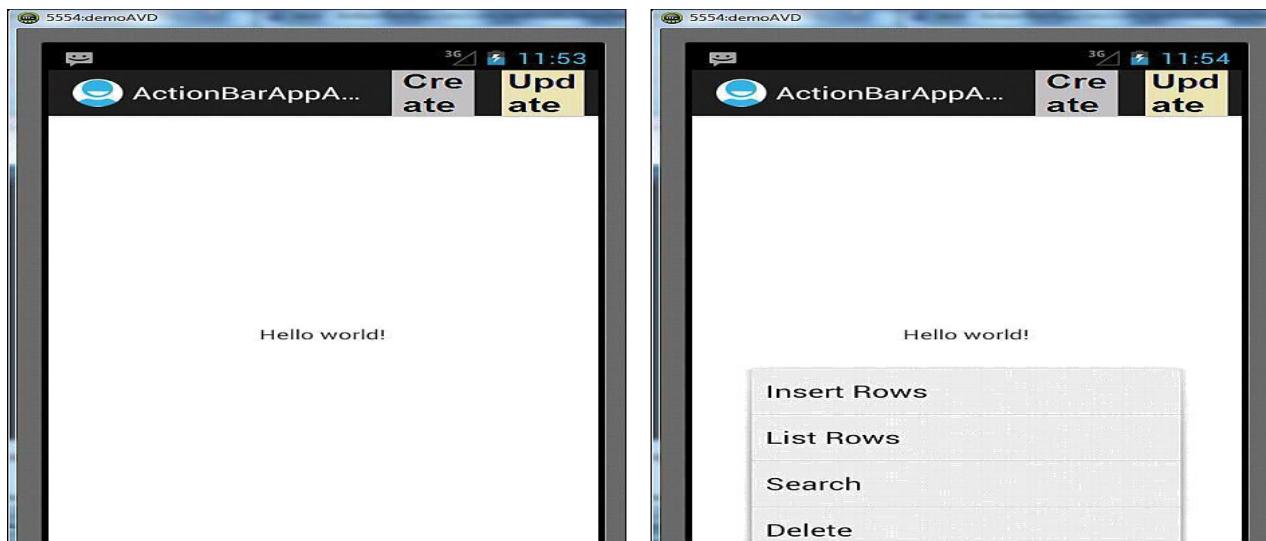


Figure. Screen on startup in portrait mode (left), and hidden actions, that is, Overflow Menu displayed (right)

On switching the device to landscape mode, we find that instead of two, four menu items appear in the ActionBar. Also, because there is enough space, the text of the menu items also appears along with the icons as shown in Figure (left). Only two menu items of our menu could not be accommodated in the ActionBar, and hence they appear in the Overflow Menu as shown in Figure (right).

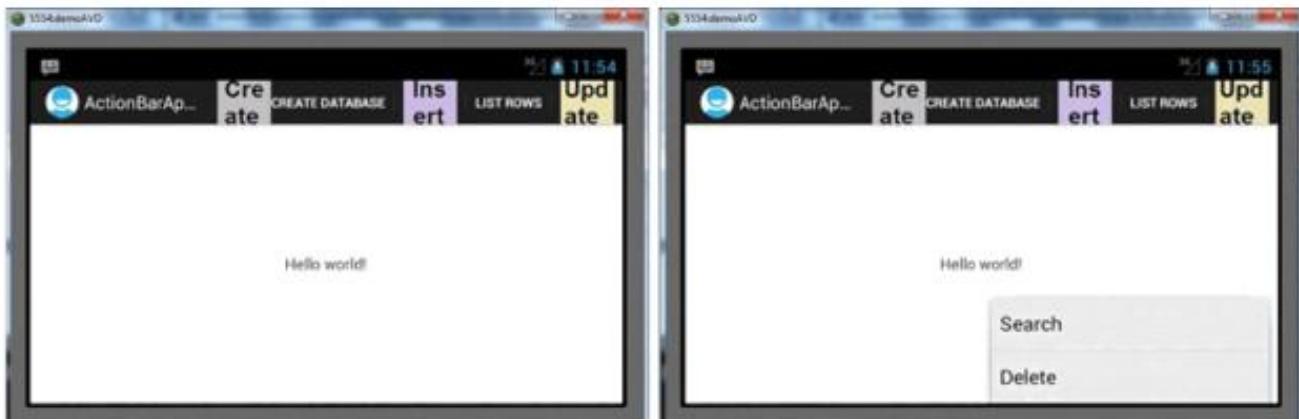


Figure. ActionBar in landscape mode (left), and Overflow Menu displayed (right)

To display checkable menu items and submenus, and to apply shortcuts to the menu items in the same way as we did in the MenuApp application, we write the code as shown in Listing in the activity_action_bar_app.xml. When compared with the code in Listing only the code in bold is new; the rest is the same as in Listing.

Listing. Code Written into the Menu File activity_action_bar_app.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    > <item android:id="@+id/create_database" android:title="Create Database"
        android:icon="@drawable/create" android:showAsAction="ifRoom|withText"
        />
    <item android:id="@+id/insert_rows" android:title="Insert Rows"
        android:icon="@drawable/insert" android:showAsAction="ifRoom" />
    <item android:id="@+id/list_rows" android:title="List Rows"
        android:showAsAction="ifRoom" />
    <item android:id="@+id/search_row" android:title="Search"
        android:icon="@drawable/search"
        android:showAsAction="ifRoom|withText" >
        <menu>
            <group android:checkableBehavior="single">
                <item android:id="@+id/search_code" android:title="Search on Code"
                    android:checked="true" />
                <item android:id="@+id/search_name" android:title="Search on Name"
                    android:alphabeticShortcut="n"
                    android:numericShortcut="6" />
                <item android:id="@+id/search_price" android:title="Search on Price" />
            </group>
        </menu>
```

```

</item>
<item android:id="@+id/delete_row" android:title="Delete"
    android:showAsAction="never" android:alphabeticShortcut="d"
    android:checkable="true" />
<item android:id="@+id/update_row" android:title="Update"
    android:icon="@drawable/update" android:showAsAction="always"
    android:alphabeticShortcut="u" android:numericShortcut="4" />
</menu>

```

We can see that the preceding code includes the android:showAsAction attribute to display the menu items in the ActionBar if the space permits. Also the <menu> element defined inside the Searchmenu item defines the submenu consisting of three menu items: Search on Code, Search on Name, and Search on Price. The Delete menu item appears as a checkable menu item, as the android:checkable attribute is set to the Boolean value true for this menu item. The android:alphabeticShortcut and the android:numericShortcut define shortcut keys of the menu items. The android:alphabeticShortcut attribute defines the shortcut key for the full keyboard, whereas the android:numericShortcut attribute defines the shortcut key from the numeric keypad.

To show a response when a menu item is selected, we need to define a TextView control in the layout file activity_action_bar_app.xml. After we define the TextView control, the layout file activity_action_bar_app.xml appears as shown in Listing

Listing. Code Written into the Layout File activity_action_bar_app.xml

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/selectedopt" />
</LinearLayout>

```

To identify the TextView in Java code, it is assigned the ID selectedopt. To display the response when a menu item is selected, modify the file ActionBarAppActivity.java to appear as shown in Listing.

Listing. Code Written into the Java Activity File ActionBarAppActivity.java

```

package com.androidunleashed.actionbarapp; import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

```

```
import android.view.MenuInflater; import android.view.MenuItem; import
android.widget.TextView;

public class ActionBarAppActivity extends Activity
{ private TextView selectedOpt;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_action_bar_app);
    selectedOpt=(TextView)findViewById(R.id.selectedopt);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) { MenuInflater inflater =
getMenuInflater(); inflater.inflate(R.menu.activity_action_bar_app, menu); return
true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{ switch (item.getItemId()) { case R.id.create_datab:
    selectedOpt.setText("You have selected Create Database option"); break;
case R.id.insert_rows:
    selectedOpt.setText("You have selected Insert Rows option"); break;
case R.id.list_rows:
    selectedOpt.setText("You have selected List Rows option"); break;
case R.id.search_row:
    selectedOpt.setText("You have selected Search Row option"); break;
case R.id.delete_row:
    selectedOpt.setText("You have selected Delete Row option"); break;
case R.id.update_row:
    selectedOpt.setText("You have selected Update Row option"); break;
case R.id.search_code:
    selectedOpt.setText("You have selected Search on Code option"); break;
case R.id.search_name:
    selectedOpt.setText("You have selected Search on Name option"); break;
case R.id.search_price:
    selectedOpt.setText("You have selected Search on Price option"); break;
} return true; } }
```

On running the application, we find the two menu items Create and Update appear as action items in the ActionBar (see Figure—left). When we select the Create action item, the TextView displays the message; you have selected Create Database option. The Overflow Menu appears as shown in Figure (middle). You can observe that the Delete menu item appears as a checkable menu item. Again, when we select any menu item from the Overflow Menu, the respective text message appears through the TextView. When we select the Search menu item from the Overflow Menu, a Search submenu appears as shown in Figure (right). The TextView confirms selection of the Search menu item by displaying the message You have selected Search Row option.

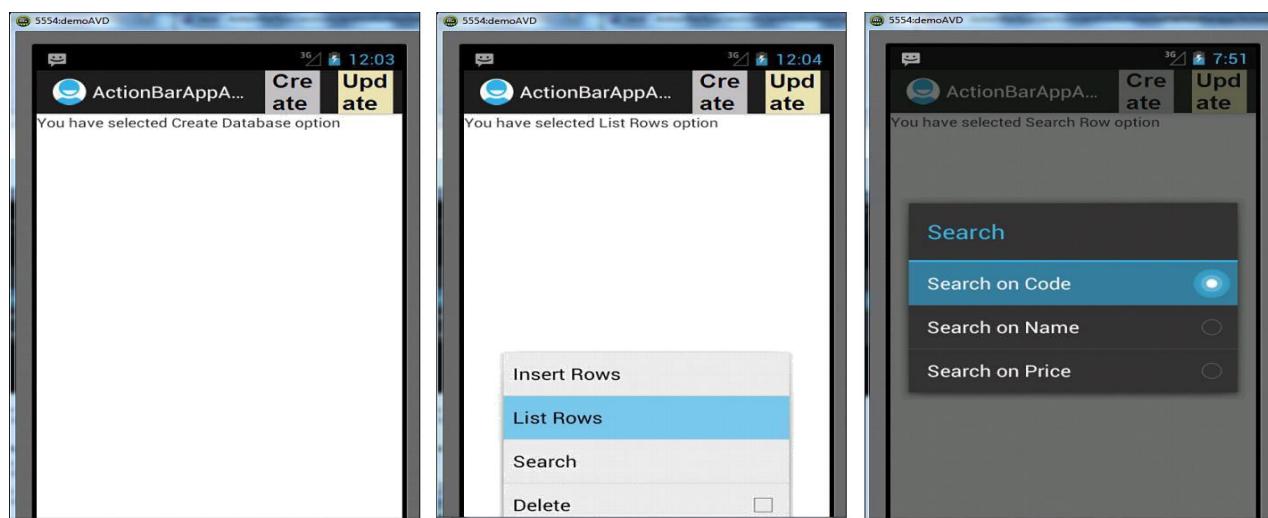


Figure. Message displayed on selecting the Create icon (left), message displayed on selecting the List Rows option (middle), and a submenu opens on selecting the Search menu item (right).

We know that the ActionBar appears in three forms: standard, tabbed, and list. The ActionBars that we have been working with until now are the standard form. Let's now learn about the tabbed ActionBar.

CREATING A TABBED ACTIONBAR

Tabbed and drop-down list ActionBars display menu items in the form of tabs and drop-down lists, respectively. They are popularly used for fragment transitions within an Activity. In tabbed and drop-down ActionBars, only one type of navigation can be enabled at a time. To display navigation tabs in an ActionBar, its `setNavigationMode()` method is called, passing the value `ActionBar.NAVIGATION_MODE_TABS` as a parameter as follows:

```
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
```

After we determine the navigation mode, we add the tabs to the ActionBar by calling its addTab()method:

```
actionBar.addTab(actionBar.newTab().setText("Create").setTabListener(this));
```

The preceding code creates a new tab, sets its text to Create, attaches a TabListener to it, and finally adds the newly created tab to the ActionBar. Just as the setText() method used in the preceding code sets the text of the tab, we can also call the setIcon() method to define an image for the tab. Besides this, we can also call the setContentDescription() method to supply more detailed information of the tab.

Example:

```
Tab tab1 = actionBar.newTab();
tabOne.setText("Create")
.setIcon(R.drawable.ic_launcher)
.setContentDescription("Creating the Database")
.setTabListener(this));
actionBar.addTab(tab1);
```

The preceding code adds a tab with the text Create to the tabbed ActionBar. The icon assigned to the tab is the default icon ic_launcher, and a detailed description assigned is “Creating the Database” to inform the user about the purpose of the tab. When we click a tab, the event is handled by the TabListener that performs the desired task.

We can better understand the concept of the tabbed ActionBar by a running example. So, create a new Android project called TabbedActionBarApp. In this application, we create two tabs, Create and Update. When either tab is selected, a respective log message is displayed. In the Java Activity file of the application, TabbedActionBarAppActivity.java, write the code as shown in Listing.

Listing. Code Written into the Java Activity File

TabbedActionBarAppActivity.java

```
package com.androidunleashed.tabbedactionbarapp; import android.app.Activity;
import android.os.Bundle;
import android.app.ActionBar; import android.app.ActionBar.Tab;
import android.app.FragmentTransaction; import android.util.Log;

public class TabbedActionBarAppActivity extends Activity implements
ActionBar.Tab- Listener {
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    final ActionBar actionBar = getSupportActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
    actionBar.setDisplayShowTitleEnabled(false);
    actionBar.addTab(actionBar.newTab().setText("Create").setTabListener(this));
    actionBar.addTab(actionBar.newTab().setText("Update").setTabListener(this));
}

@Override
public void onTabReselected(Tab tab, FragmentTransaction ft) { Log.d("Tab",
    String.valueOf(tab.getPosition()) + " re-selected");
}

@Override
public void onTabSelected(Tab tab, FragmentTransaction ft) { Log.d("Tab",
    String.valueOf(tab.getPosition()) + " selected");
}

@Override
public void onTabUnselected(Tab tab, FragmentTransaction ft) { Log.d("Tab",
    String.valueOf(tab.getPosition()) + " Unselected");
}

```

In the preceding code, we can see that an `ActionBar` object, `actionBar`, is created by calling the `getActionBar()` method. To make the `ActionBar` appear in the form of tabs, its navigation mode is set to `ActionBar.NAVIGATION_MODE_TABS`. The Activity title is made invisible by passing the Boolean value `false` to the `setDisplayShowTitleEnabled()` method. Thereafter, two tabs with the text Create and Update are respectively created and added to the `ActionBar`. The event listener `TabListener` is associated with both the tabs. When either tab is selected, the `onTabSelected()` method is invoked, and the selected tab is passed to it as a parameter. The `onTabSelected()` method displays the log message informing the position of the selected tab. The position of the tab is zero numbered; that is, the first tab is considered to have the position 0, the second tab has the position 1, and so on. When a tab is selected, the `onTabUnselected()` method is also called, and the other tab that is not selected is passed to it as the parameter. The `onTabUnselected()` method displays the

position of the other tab that is not selected. On running the application, we find the tabbed ActionBar showing two tabs as shown in Figure (left). When we select the first tab, Create, the log messages 1 Unselected and 0 selected are displayed confirming that the first tab, Create, is selected and the second tab, Update, is unselected. Similarly, when we select the second tab, Update, the log messages 0 Unselected and 1 selected are displayed as shown in Figure (right).

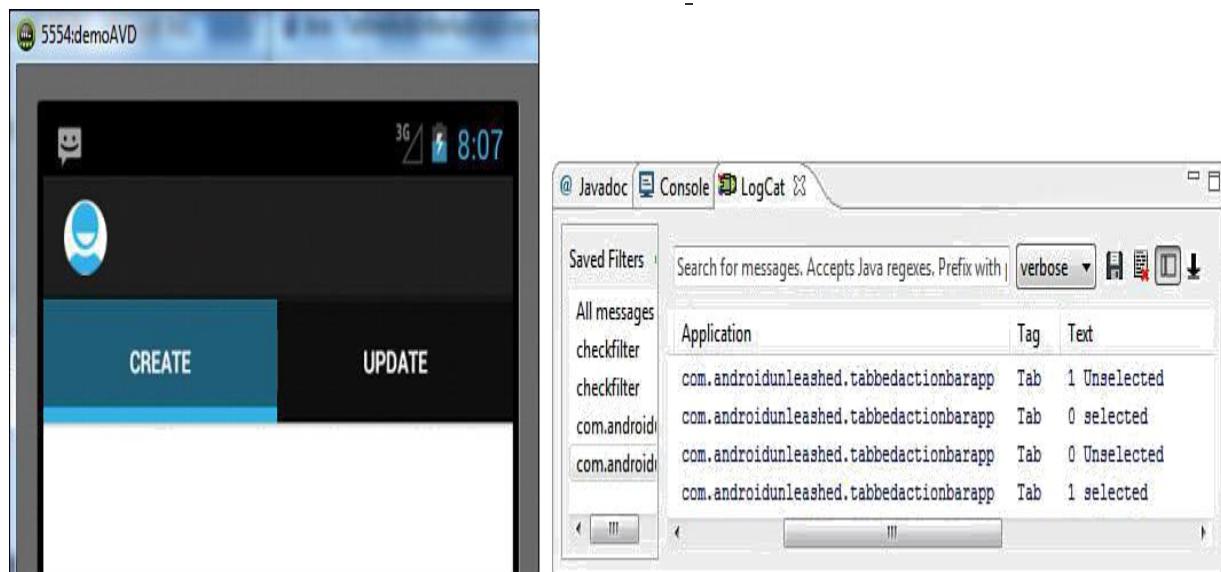


Figure. Screen showing two tabs in the tabbed ActionBar (left), and log messages displayed on selecting the action tabs (right)

After understanding the tabbed ActionBar, let's learn how the drop-down list ActionBar is created.

6.7 CREATING A DROP-DOWN LIST ACTIONBAR

In a drop-down list ActionBar, the menu items are displayed in the form of a drop-down list. It is popularly used for displaying the content within an Activity on the basis of the selection made by the user. To display a drop-down list in an ActionBar, its `setNavigationMode()` method is called, passing the value `ActionBar.NAVIGATION_MODE_LIST` as a parameter to it as shown here:

```
ActionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
```

The drop-down list as expected appears like a spinner, displaying a list of available options, allowing us to select one of them. For displaying options in the drop-down list,

Remember that an ArrayAdapter is the simplest of the adapters and acts as the data source for the selection widgets ListView, GridView, and so on. First, we define a string array containing the strings that we want to be displayed in the drop-down list. Thereafter, we create an ArrayAdapter that displays the elements of the array in the form of drop-down items. That is, the elements of the array are wrapped or cast into the spinner drop-down items. Finally, the ArrayAdapter is assigned to the ActionBar for displaying the menu items, that is, array elements in the form of drop-down menu items. To assign the ArrayAdapter to the ActionBar and to attach the event listener to the drop-down items that are displayed, the setListNavigationCallbacks() method is called, passing the adapter and OnNavigationListener to it as parameters as shown in the following code:

```
String[] items = new String[] { "Create", "Insert", "Update", "Search" };
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_spinner_dropdown_item, items);
ActionBar actionBar = getActionBar();
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
actionBar.setListNavigationCallbacks(adapter, onNavigationItemSelected);
```

the actionBar passing the ArrayAdapter, adapter, and the listener, onNavigationSelected, to it as parameters. That is, we assign the callbacks to handle drop-down selections. When a user selects an item from the drop-down list, the onNavigationItemSelectedhandler is called where we can write the code to perform the desired action.

Let's create a drop-down list ActionBar in an Android project. Create a new Android project called ListActionBarApp. In this application, we display a few menu items in the form of a drop-down list, and when any menu item is selected, a respective log message is displayed. In the Java Activity file of the application ListActionBarAppActivity.java, write the code as shown in Listing.

Listing. Code Written into the Java Activity File ListActionBarAppActivity.java

```
package com.androidunleashed.listactionbarapp;

import android.app.Activity; import android.os.Bundle; import
android.app.ActionBar.OnNavigationListener;
import android.app.ActionBar; import android.widget.ArrayAdapter;
import android.util.Log;

public class ListActionBarAppActivity extends Activity { @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

String[] items = new String[] { "Create", "Insert", "Update", "Search"
}; ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_dropdown_item, items);
ActionBar actionBar = getActionBar();
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
actionBar.setListNavigationCallbacks(adapter, onNavigationItemSelected);
}
OnNavigationListener onNavigationItemSelected = new OnNavigationListener()
{
@Override
public boolean onNavigationItemSelected(int itemPosition, long itemId)
{
Log.d("Option ", String.valueOf(itemId) + " is selected"); return true;
}
};
}
}

```

In the preceding code, we notice that when an item from the drop-down list is selected, the `onNavigationItemSelected()` method is called. The parameters `itemPosition` and `itemId` in the `onNavigationItemSelected()` method contain the information about the position and ID of the selected item. A log message is displayed in this method displaying the ID of the selected item. The IDs are sequentially assigned to the items in the drop-down list beginning with 0. To enable the ActionBar, don't forget to set the value of the `android:minSdkVersion` attribute to 11 or higher in the `AndroidManifest.xml` file.

On running the application, we get a Spinner as shown in Figure (top left). The Spinner shows the first item of the drop-down list, Create. The default style shows the first item in a dark color, which is almost invisible in the dark background color. Open the `styles.xml` file from the `res/values` folder and add a custom style called `MyActionBar` to it through the following statement:

```

<style name="MyActionBar"
parent="@android:style/Widget.Holo.Light.ActionBar" />

```

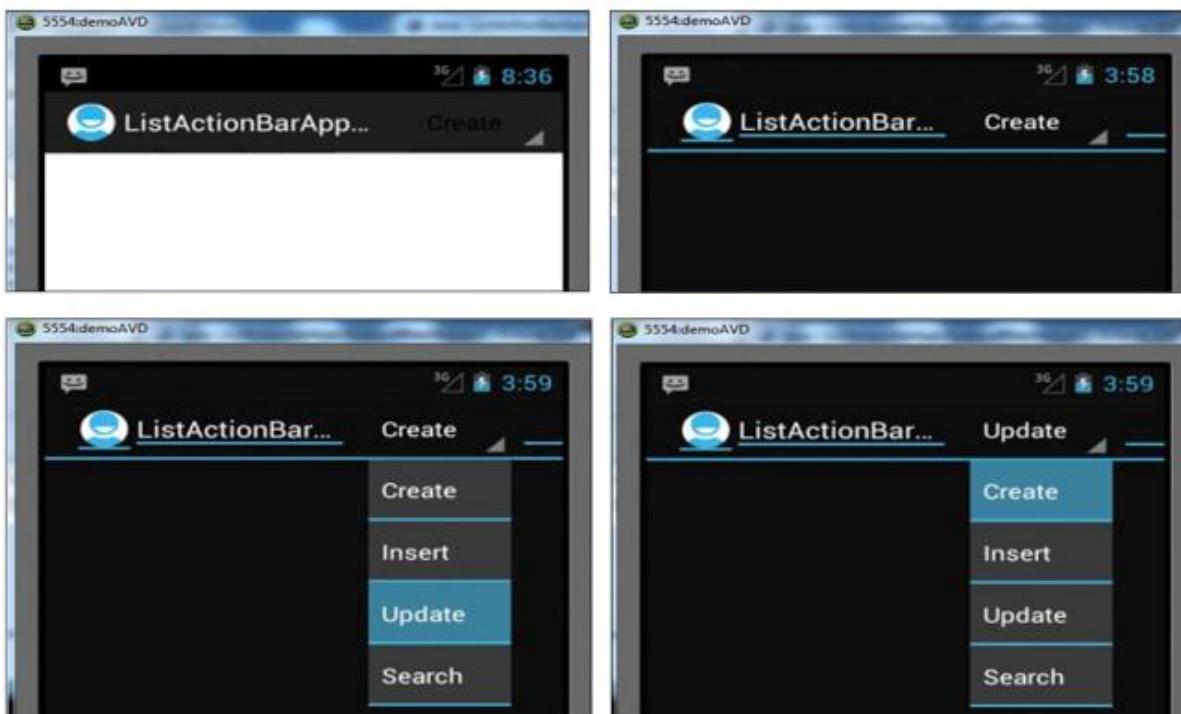


Figure. First Item of the spinner almost invisible (top left), first item of the spinner, Create, becomes visible (top right), all actions displayed on selecting the list (bottom left), and selected list item displayed at the header of the list (bottom right)

After we add this style, the styles.xml file appears as shown in Listing.

Listing. Code Written into the styles.xml File

```
<resources xmlns:android="http://schemas.android.com/apk/res/android">
<style name="AppTheme" parent="android:Theme.Light" />
<style name="MyActionBar"
parent="@android:style/Widget.Holo.Light.ActionBar" />
</resources>
```

To implement the preceding style to our application, open the AndroidManifest.xml file and set the value of android:theme attribute as shown here:

android:theme="@style/MyActionBar"

The preceding statement applies the MyActionBar style to our application. The output now appears as shown in Figure (top right).

When we select the Spinner, the drop-down list opens showing all the available items as shown in Figure (bottom left). When we select an item, Update, it appears as the Spinner's header as shown in Figure_(bottom right), informing that it was selected in

the previous selection. Figure shows the log messages displayed on selecting the two items Update and Create from the drop-down list. The ID of the selected drop-down item is displayed using the itemId parameter in the onNavigationItemSelected() method.

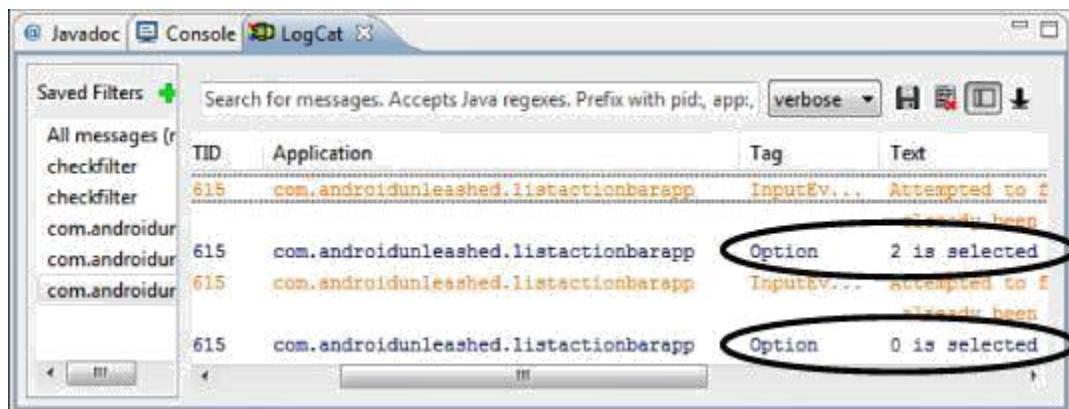


Figure Log messages displayed on selecting the actions from the list

ActionBar SUMMARY

In this chapter, we learned about different types of menus and saw how to create menus through XML, as well as by coding. We learned to create Options Menus, define Icon Menus, and handle menu selections. We saw how to define Expanded Menus, add Submenus, create Context Menus, and handle Context Menu selections. We learned to use check boxes and radio buttons in menus, add shortcut keys, and apply Context Menus to ListViews. We learned to create instant access menu items through ActionBars and also saw the procedure of creating tabbed and drop-down ActionBars.

CHAPTER 7

TELEPHONY OPERATIONS

7.1 UNDERSTANDING BROADCAST RECEIVERS

A broadcast receiver is a component that responds to different messages that are broadcast by the system or other applications. Messages such as new mail, low battery, captured photo, or completed download require attention. Such messages are broadcast as an Intent object that is waiting for broadcast receivers to respond. The broadcast receiver responds to such broadcasted Intents and takes the necessary actions. The broadcast receivers can, for example, create a status bar notification to alert the user when a broadcast occurs.

Note

A broadcast Intent can invoke more than one receiver.

We cover two separate aspects of broadcast receivers:

- Broadcasting an Intent
- Receiving the broadcast Intent

Broadcasting an Intent

To broadcast an Intent, we first create an Intent object, assign a specific action to it, attach data or a message to the broadcast receiver, and finally broadcast it. We can optionally put an extra message or data on the Intent. Table lists the methods involved in broadcasting an Intent.

Table. Methods Involved in Broadcasting an Intent

Method	Description
<code>putExtra()</code>	Used to add data or a message to the Intent that we want to send to the broadcast receiver. Syntax: <code>putExtra(String name, String value)</code> Where <code>name</code> is the key or name of the value that we want to pass along with the Intent. The name is used to identify the value.
<code>setAction()</code>	Used to set the action to perform on the data or message being sent with the Intent. Syntax: <code>setAction(String action)</code> The broadcast receiver uses the <code>getAction()</code> method to retrieve the action to be performed on the received data.
<code>sendBroadcast()</code>	Available on the Context class, this method is used to send the broadcast Intent to all the registered Intent receivers. Syntax: <code>void sendBroadcast(Intent intent_to_broadcast)</code> Where the <code>intent_to_broadcast</code> parameter represents the Intent that we want to broadcast.

The following code broadcasts an Intent:

```
public static String BROADCAST_STRING =
"com.androidunleashed.testingbroadcast"; Intent broadcastIntent = new
Intent();
broadcastIntent.putExtra("message", "New Email arrived");
broadcastIntent.setAction(BROADCAST_STRING);
sendBroadcast(broadcastIntent);
```

We can see that an Intent object called broadcastIntent is created. The data or message to be passed along with the Intent is "New Email arrived", and the name or key assigned to this message is message. The action string is made unique by using a namespace similar to a Java class. We can see that the com.androidunleashed.testingbroadcast is assigned as an action to the Intent. Finally, the broadcastIntent is sent or broadcasted and received by the broadcast receivers.

Receiving the Broadcast Intent

A broadcast receiver is a class that extends the BroadcastReceiver. It also needs to be registered as a receiver in an Android application via the AndroidManifest.xml file or through code at runtime. The broadcast receiver class needs to implement the onReceive() method. The following is sample code of an onReceive() method:

```
public void onReceive(Context context, Intent intent) { String actionName =
intent.getAction(); if(actionName != null &&
actionName.equals("com.androidunleashed.testingbroad-cast")) {
    String msg = intent.getStringExtra("message");
    Log.d("Received Message: ",msg);
}
}
```

The getAction() and getStringExtra() methods used here need some explanation:

- **getAction()**—Retrieves the action to be performed from the Intent object. It is the action that indicates what task has to be performed on the data passed along with the Intent.

Syntax:

getAction()

- **getStringExtra()**—Retrieves the extended data from the Intent.

Syntax:

getStringExtra(String name)

where name represents the key or the name assigned to the value while adding data to the Intent through the `putExtra()` method. In the `onReceive()` method, we access the Intent object passed as a parameter. From the Intent object, we retrieve the action that is supposed to be performed. If the action to be performed is not null and matches the one that is sent by the sender activity, the message or data passed along with the Intent is extracted from the Intent and is logged.

Let's create an Android project called BroadcastApp. In this application, a Button is displayed with the caption Send Broadcast. After we click the Button, an Intent with a message is broadcast. The broadcasted Intent is then received through a broadcast receiver, and the message sent with the Intent is extracted and logged. To define a Button control, write the code shown in [Listing 11.1](#) into the `activity_broadcast_app.xml` layout file.

Listing 11.1. Code Written into `activity_broadcast_app.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button android:id="@+id/broadcast_button" android:text="Send Broadcast"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_gravity="center" />
</LinearLayout>
```

We can see that the Button control is assigned the `broadcast_button` ID, which is used to identify it in the Java code. The caption assigned to the Button control is Send Broadcast. Next, we need to write code in the Java activity file to define an Intent object, assign action and add data to it, and then broadcast it. To do so, the code shown in Listing 11.2 is written into `BroadcastAppActivity.java`.

Listing 11.2. Code Written into `BroadcastAppActivity.java`

```
package com.androidunleashed.broadcastapp;
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.widget.Button;
import android.view.View;
```

```

public class BroadcastAppActivity extends Activity {
    public static String BROADCAST_STRING =
    "com.androidunleashed.testingbroadcast"; @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_broadcast_app);
        Button broadcastButton = (Button) this.findViewById(R.id.broadcast_button);
        broadcastButton.setOnClickListener(new Button.OnClickListener(){
            public void onClick(View v) {
                Intent broadcastIntent = new Intent();
                broadcastIntent.putExtra("message", "New Email arrived");
                broadcastIntent.setAction(BROADCAST_STRING);
                sendBroadcast(broadcastIntent);
            }
        });
    ;
    }
}

```

Here we can see that the Button control with the broadcast_button ID is accessed from the layout file and is mapped to the Button object broadcastButton. A ClickListener is associated with the Button control. When the Button control is clicked, the callback method onClick() is invoked. In the onClick() method, an Intent object called broadcastIntent is defined. A message, New Email arrived, is added to the broadcastIntent object with the key message. With the help of a static string, BROADCAST_STRING, a unique action, com.androidunleashed.testingbroadcastis assigned to the Intent object, broadcastIntent. Finally, the Intent is broadcasted by calling the sendBroadcast() method.

The next step is defining an Activity that acts as a broadcast receiver. So, to the package com.androidunleashed.broadcastapp of our application, add a Java file, ReceiveBroadcastActivity.java. To respond to the broadcasted Intent and to access the data passed along with it, write the code shown in Listing in the Java file ReceiveBroadcastActivity.java.

Listing. Code Written into ReceiveBroadcastActivity.java

```

package com.androidunleashed.broadcastapp;
import android.content.BroadcastReceiver;
import android.content.Intent;

```

```

import android.content.Context;
import android.util.Log;

public class ReceiveBroadcastActivity extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        String actionName = intent.getAction();
        if(actionName != null && actionName.equals("com.androidunleashed.testingbroadcast")) {
            String msg = intent.getStringExtra("message");
            Log.d("Received Message: ",msg);
        }
    }
}

```

The ReceiveBroadcastActivity.java file, which is the broadcast receiver, has to be registered in the manifest file. The code for registering the activity is as follows:

```

<receiver android:name=".ReceiveBroadcastActivity">
    <intent-filter>
        <action android:name="com.androidunleashed.testingbroadcast"></action>
    </intent-filter>
</receiver>

```

We can see that the `<receiver>` tag is used in the manifest file to register the broadcast receiver. The tag also designates the `ReceiveBroadcastActivity.class` as the recipient of the Intent whose action is `com.androidunleashed.testingbroadcast`. Listing shows the code in the `AndroidManifest.xml` file. Only the code in bold is added; the rest is the default code that is auto-generated by the Android SDK.

Listing. Code in the `AndroidManifest.xml` File

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.broadcastapp" android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />
    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity android:name=".BroadcastAppActivity"
            android:label="@string/title_activity_broadcast_app"
        > <intent-filter>

```

```

<action android:name="android.intent.action.MAIN" /> <category
    android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<receiver android:name=".ReceiveBroadcastActivity">
    <intent-filter>
        <action android:name="com.androidunleashed.testingbroadcast"></
action>
    </intent-filter>
</receiver>
</application>
</manifest>

```

After running the application, we see a Button with the caption Send Broadcast displayed on the screen, as shown in Figure (left). After we click Send Broadcast, an Intent with the message New Email arrived is broadcast. The ReceiveBroadcastActivity.class receives the broadcasted Intent and extracts the message, New Email arrives from it and logs it. The logged message appears in the LogCat window, as shown in Figure (right).

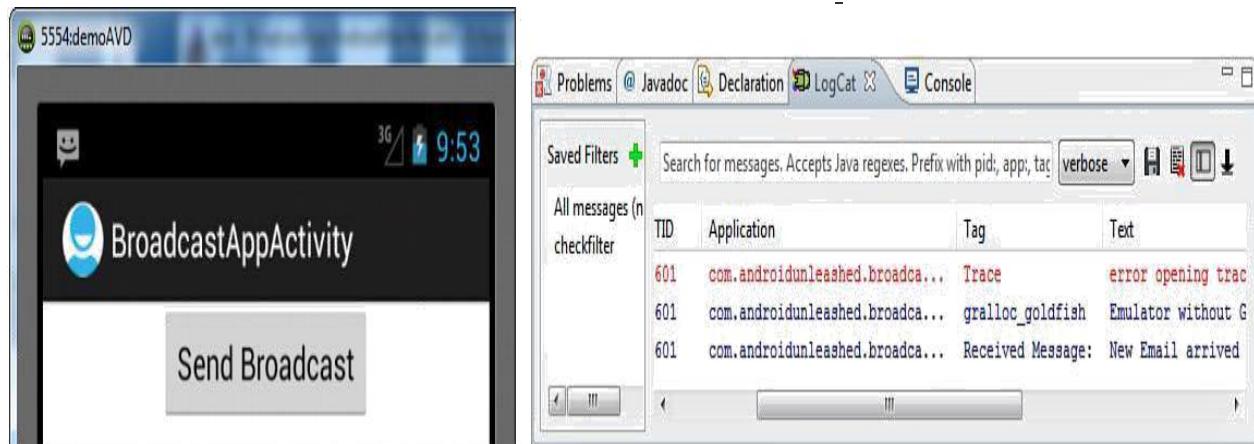


Figure. Application showing the Send Broadcast button on startup (left), and logged messages displayed in the LogCat window (right)

Note

We can have more than one receiver receive the broadcasted Intent.

7.2 USING THE NOTIFICATION SYSTEM

The Android notification system provides us with several ways of alerting users. For

example, the user can be notified with text, vibration, blinking lights, and sound indicators. Notifications are usually displayed on the status bar at the top of the screen.

Notification via the Status Bar

The simplest type of notification is status, which appears in the status bar as an icon along with some optional ticker text. Users can pull down the status bar to see the notification list and clear the notification by clicking the Clear button. After tapping the notification, the user navigates to the Intent defined by the notification. The status notification never launches an activity automatically, but simply notifies the user and launches the activity only when the notification is selected. Besides an icon and

Note

The ticker text is briefly displayed in the status bar when the notification fires.

ticker text, the notification can have a title and body text displayed when the full notification is being displayed.

For creating notifications, the following two classes are used:

- **Notification**—The object that defines the information to be displayed, which can be text to be displayed on the status/expanded status bar, an icon displayed with the text, the number of times the notification is triggered, and so on.
- **NotificationManager**—The base object with which notifications are handled. It displays the information encapsulated in the Notification object, which is displayed via the notify() method.

Creating Notifications

The first step is to create a Notification object and configure it by defining notification properties. The following code shows how to do so:

```
Notification notification = new Notification();
notification.icon = R.drawable.ic_launcher;
notification.tickerText = "There is a new notification";
notification.when = System.currentTimeMillis();
notification.flags |= Notification.FLAG_AUTO_CANCEL;
```

Here, we see that a Notification object called notification is created and thereafter its public members are used to configure it:

- **icon**—Assigns the notification icon.
- **tickerText**—Assigns the small notification text.
- **when**—Assigns the time when the notification occurred. We use the system time to specify the time the notification occurred.

- **flag**—Assigns the constant that determines the subsequent action when the notification is selected from the notification window. We usually assign the FLAG_AUTO_CANCEL constant to this public variable, which specifies that the notification be automatically canceled after it is selected from the notifications.

We can also assign a notification icon, ticker text, and time of occurrence through the Notificationobject constructor, as shown here:

```
Notification notification = new Notification(R.drawable.ic_launcher, "There is a new notification", System.currentTimeMillis());
```

Creating PendingIntent

After receiving the notification, we may choose to take a necessary action. We use the PendingIntentclass to switch to the desired Intent when the notification is tapped.

The PendingIntent class enables us to create Intents that can be triggered by our application when an event occurs. The following code creates a PendingIntent called pendIntent:

```
Intent intent = new Intent(getApplicationContext(), TargetActivity.class);  
PendingIntent pendIntent = PendingIntent.getActivity(getApplicationContext(), 0,  
intent, 0);
```

We can see that we create an Intent object by supplying the current application context and the activity name TargetActivity.class—the one that we want to launch. Thereafter, a PendingIntent object called pendIntent is created by supplying the following four parameters to the getActivity() method:

- The current application context in which the PendingIntent starts the activity.
- A request code for the sender. Because it is not used, we supply a value of 0 for this parameter.
- The Intent of the activity to be launched. We supply the Intent object for this parameter.
- Flags to specify the unspecified parts of the Intent to be sent. We supply a value of 0 for this parameter, as there is no unspecified part.

To display text after expanding the notification, and to specify the pending Intent that we want to launch, the setLatestEventInfo() method of the Notification class is used. The method is now deprecated. The syntax of the method is

```
setLatestEventInfo(application_context, title, text, pending_intent);
```

where application_context represents the current application context, title represents the title of the notification, and text represents the notification text displayed after expanding the status bar. The pending_intent represents the PendingIntent object to

supply the activity information we want to launch when a notification is tapped.

Example:

```
notification.setLatestEventInfo(getApplicationContext(), "New E-mail", "You have one unread message.", pendingIntent);
```

The `setLatestEventInfo()` method is deprecated. The task of configuring a notification is done through `Notification.Builder`.

Instead of using the method shown here, we can use the `Notification.Builder`.

Using `Notification.Builder`

`Notification.Builder` is the Builder class for `Notification` objects and provides several methods to configure notification, as shown in Table

Table. `Notification.Builder` Class Methods

Method	Description
<code>setSmallIcon()</code>	Used to supply the small icon resource that is displayed to represent the notification in the status bar. Syntax: <code>setSmallIcon(int icon)</code> where the <code>icon</code> parameter represents the resource ID of the drawable to be used as the icon of the notification.
<code>setAutoCancel()</code>	Used to determine whether we want to make the notification invisible when it is tapped. The Boolean value <code>true</code> is supplied to this method to make the notification invisible. Syntax: <code>setAutoCancel(boolean autoCancel)</code>
<code>setTicker()</code>	Used to supply the ticker text that is displayed in the status bar when the notification arrives. Syntax: <code>setTicker(CharSequence textMessage)</code>
<code>setWhen()</code>	Used to supply the time of occurrence of the notification. Syntax: <code>setWhen(long timeOfOccurrence)</code>
<code>setContentTitle()</code>	Used to supply the title of the notification when the status bar is expanded. Syntax: <code>setContentTitle(CharSequence title)</code>
<code>setContentText()</code>	Used to supply the text of the notification. Syntax: <code>setContentText(CharSequence text)</code>
<code>setContentIntent()</code>	Used to supply a <code>PendingIntent</code> to be sent when the notification is tapped. Syntax: <code>setContentIntent(PendingIntent intent)</code>

The following code shows how to use the `Notification.Builder` methods shown in Table to configure the notification:

```
Notification.Builder builder = new Notification.Builder(getApplicationContext())
.setSmallIcon(R.drawable.ic_launcher)
.setAutoCancel(true)
.setTicker("There is a new notification")
.setWhen(System.currentTimeMillis())
.setContentTitle("New E-mail")
.setContentText("You have one unread message.")
.setContentIntent(pendingIntent);
notification = builder.getNotification();
```

The preceding code configures a notification as shown here:

- Sets the ic_launcher.png image as the notification icon
- Makes the notification invisible when tapped
- Assigns the text There is a new notification as the ticker text of the notification
- Sets the current time as the time of occurrence of the notification
- Sets the title of the notification as New E-mail
- Sets the body text of the notification as You have one unread message
- Fires the specified pending Intent, pendIntent, when the notification is tapped

The configuration notification created through the Notification.Builder object is assigned to the Notification object called notification.

Obtaining a NotificationManager

The NotificationManager class executes and manages all status notifications. To obtain a valid NotificationManager, we use the getSystemService() method:

```
NotificationManager notificationManager =
(NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
```

After obtaining the NotificationManager object, we can invoke its notify() method to notify the user by displaying the notification.

Syntax:

```
notify(uniqueID, notificationObject)
```

where the uniqueID parameter represents the application unique identifier, and the notificationObject parameter represents the Notification object that we want to display.

Example:

```
notificationManager.notify(0, notification);
```

Let's apply what you've learned so far and create an application that sends notifications to the user. Create a new Android project called NotificationApp. In this application, we create a Button with the caption Create Notification. When clicked, the Button creates a notification that displays ticker text in the status bar. When the user expands the status bar, the notification's title and body text are displayed. After we tap the notification, the program jumps to an activity that displays a message indicating that a new activity is launched.

Listing. Code Written into activity_notification_app.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/createbutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Create Notification" android:layout_gravity="center" />
</LinearLayout>
```

We can see that the Button control is assigned the createbutton ID and a caption called Create Notification. The ID of the Button control identifies it in the Java code.

Because we want to launch an activity when the notification is tapped, let's define the layout file for the new activity. Add an XML file called target.xml to the res/layout folder. Through the activity that is launched, we need to display a text message through the TextView control. To define the TextViewcontrol in a LinearLayout container, write the code shown in [Listing 11.6](#) into the target.xml file.

Listing 11.6. Code Written into target.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/messageview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is target activity"
        android:layout_gravity="center" />
</LinearLayout>
```

We can see that the TextView control is assigned the messageview ID, set to display This is target activity, and aligned to appear at the center of the View.

For the new Activity, add a Java file to the com.androidunleashed.notificationapp package of our project. Assign the name TargetActivity.java to the newly added Java file. The new Activity has nothing to do but display the TextView defined in its layout file target.xml. Write the code shown in [Listing 11.7](#) into the Java file TargetActivity.java.

Listing 11.7. Code Written into TargetActivity.java

```
package com.androidunleashed.notificationapp;
import android.app.Activity; import android.os.Bundle;

public class TargetActivity
    extends Activity { @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.target);
    }
}
```

We can see that target.xml is set as the ContentView of the new Activity TargetActivity.java. Next, we need to write the code in the NotificationAppActivity.javamain activity file to perform the following tasks:

- Create a Notification object and configure it to display an icon, title, and text.
- Create a PendingIntent to launch the activity when the notification is tapped.
- Create a NotificationManager object to display and manage the notification.

Listing. Code Written into NotificationAppActivity.java

```
package com.androidunleashed.notificationapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.app.Notification;
import android.widget.Button;
import android.view.View.OnClickListener;

public class NotificationAppActivity extends Activity
    { @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_notification_app);
Button createButton = (Button)
findViewById(R.id.createbutton);
createButton.setOnClickListener(new
OnClickListener() {
    @Override
    public void onClick(View arg0) {
        Intent intent = new Intent(getApplicationContext(), TargetActivity.class);
        PendingIntent pendingIntent = PendingIntent.
getActivity(getApplicationContext(), 0, intent, 0);
        NotificationManager notificationManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
        Notification notification = new Notification();
        Notification.Builder builder = new Notification.
Builder(getApplicationContext())
.setSmallIcon(R.drawable.ic_launcher)
.setAutoCancel(true)
.setTicker("There is a new notification")
.setWhen(System.currentTimeMillis())
```

```

.setContentTitle("New E-mail")
.setContentText("You have one unread message")
.setContentIntent(pendIntent); notification = builder.getNotification();
notificationManager.notify(0, notification);

}
);
}
}

```

The Android application never recognizes the newly added activity until it is mentioned in the `AndroidManifest.xml` configuration file. So, add the statement shown in bold to `AndroidManifest.xml`

Listing. Code Written into `AndroidManifest.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.notificationapp" android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />
    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity android:name=".NotificationAppActivity"
            android:label="@string/title_activity_notification_app" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" /> <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".TargetActivity"
            android:label="@string/app_name" />
    </application>
</manifest>

```

The Android project now can recognize the `TargetActivity.java` activity file. After we run the application, a Button control with the caption Create Notification is displayed (see Figure— top left). After we click the Create Notification button, a notification is created and its ticker text, There is a new notification, is displayed at the top of the screen in the status bar (Figure—top right). After we pull down the status bar, the title of the notification and its body text

are displayed, as shown in Figure (bottom left). We can see that the title of the notification is New E-mail, and the text reads You have one unread message. After we select the notification, a new activity is launched and is confirmed by the message This is target activity displayed via its TextView control (see Figure—bottom right).

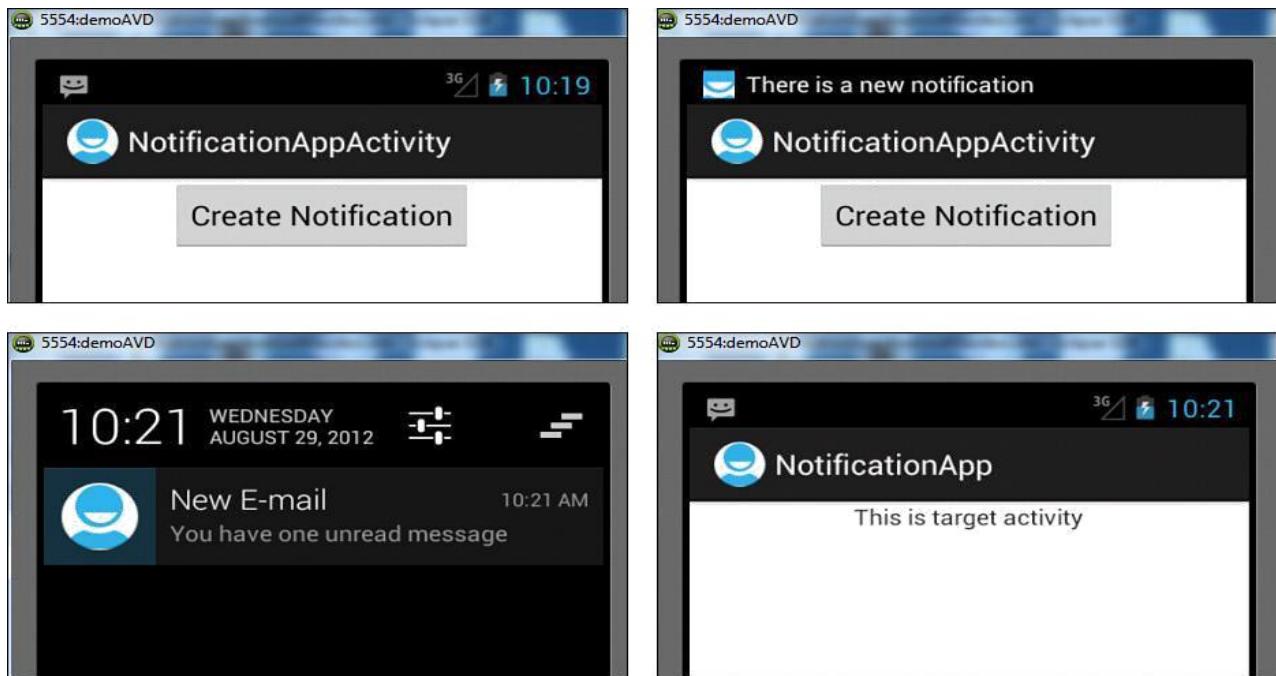


Figure. The application showing the Create Notification button on startup (top left), the ticker text of the notification (top right), the notification title and text displayed after expanding the status bar (bottom left), and the new activity launched after selecting the notification (bottom right)

7.3 SENDING SMS MESSAGES WITH JAVA CODE

Sending and receiving SMS messages is considered as one of the most economical and popular modes of communication. To implement the SMS Messaging facility in our application, Android provides a built-in class known as `SmsManager`, which we can use to send and receive SMS messages. For testing SMS messaging, we don't need a real device but can easily test it on the Android emulator.

To understand the concept, let's create a new Android project called `SendSMSApp`. The first step is to design the user interface for sending messages via SMS. The user interface consists of three `TextView`, two `EditText`, and two `Button` controls. One of the `TextView` controls is for displaying the title of the screen, Message Sending Form. The other two `TextView` controls are used on the left of the `EditText` controls to display text that tells the user what has to be entered in the `EditText` controls. These two `TextView` controls are used to display `To:` and `Message::`. The two

EditText controls are used to enter the phone number of the recipient and the message to be sent.

To define this interface, let's modify the code of the activity_send_smsapp.xml layout file to appear as shown in Listing

Listing. Code Written into activity_send_smsapp.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView android:layout_height="wrap_content" android:text="Message
        Sending Form" android:textStyle="bold" android:textSize="18sp"
        android:layout_width="match_parent" android:gravity="center_horizontal"/>
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="To:" />
    <EditText android:id="@+id/recvr_no"
        android:layout_height="wrap_content"
        android:layout_width="match_parent" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Message:" />
    <EditText android:id="@+id/txt_msg"
        android:layout_width="match_parent"
        android:layout_height="150dp" />
    <RelativeLayout android:layout_width="match_parent"
        android:layout_height="match_parent" android:orientation="horizontal">
        <Button android:id="@+id/send_button" android:text="Send SMS"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_marginLeft="40dip" android:paddingLeft="20dip"
            android:paddingRight="20dip" />
        <Button android:id="@+id/cancel_button" android:text="Cancel"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_toRightOf="@+id/send_button"
            android:layout_marginLeft="15dip" android:paddingLeft="20dip"
            android:paddingRight="20dip" />
    </RelativeLayout>
</LinearLayout>
```

The two EditText controls are to be used to enter the phone number of the recipient and the text message; hence the IDs assigned to them are recvr_no and txt_msg. Because a text message can be long, the layout_height attribute of the txt_msg

EditText control is set to 150dp. The height is large enough to enter long messages. Moreover, the text scrolls vertically within the given height to accommodate longer messages. The two Button controls are assigned the text Send SMS and Cancel to show the kind of task they perform when clicked. The Send SMS button is assigned to the send_button ID, and the Cancel button is assigned to the cancel_button ID.

Getting Permission to Send SMS Messages

To send and receive SMS messages in an application, we need to use permissions. All the permissions our application uses are defined in the AndroidManifest.xml file, so that when the application is installed on a device, the user is informed of the access permissions that the application uses. We need to add permissions to the <manifest> element of our AndroidManifest.xml file:

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Listing shows the code in the AndroidManifest.xml file. The statement in bold is the added code; the rest is the default code.

Listing. Code Written into AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.sendsmsapp" android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity android:name=".SendSMSAppActivity"
            android:label="@string/title_activity_send_smsapp"
            > <intent-filter>
                <action android:name="android.intent.action.MAIN" /> <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Writing Java Code

To add an action to the send_button and cancel_button, we need to write Java code

into the `SendSMSAppActivity.java` activity file. Before we write code, let's define what we want the two buttons to do. We want the button with the `send_button` ID to do the following tasks:

- Validate the content of the `recv_no` and `txt_msg` `EditText` controls. Recall that the two `EditText` controls are meant for entering the phone number of the recipient and text message of the SMS. If either of the two `EditText` controls is empty, we want to suspend the process and prompt the user to enter data into both the controls.
- Send an SMS message to the recipient.
- Inform the user whether the SMS message was successfully sent and delivered. The difference between SMS Sent and SMS Delivered status is that the former means the SMS message was received by the server or SMSC (Short Message Service Center). The latter means that the SMS message was received by the recipient from the server. Remember that if the recipient is out of range or offline the SMS message still can be Sent. When the recipient actually receives the message, it is then successfully delivered.

As far as the `cancel_button` is concerned, we want it to cancel the operation and delete the content, if any, in the two `EditText` controls.

Listing. Code Written into `SendSMSAppActivity.java`

```
package com.androidunleashed.sendsmsapp; import android.app.Activity;
import android.os.Bundle;
import android.telephony.SmsManager; import android.widget.Button;
import android.view.View; import android.widget.EditText;
import android.widget.Toast;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.Context; import android.content.IntentFilter;

public class SendSMSAppActivity extends Activity { EditText phoneNumber,
message; BroadcastReceiver sentReceiver, deliveredReceiver; String SENT =
"SMS_SENT"; String DELIVERED = "SMS_DELIVERED";

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_send_smsapp);
    final PendingIntent sentPendingIntent =
PendingIntent.getBroadcast(this, 0, new Intent(SENT), 0);
    final PendingIntent deliveredPendingIntent =
PendingIntent.getBroadcast(this, 0,
```

```

new Intent(DELIVERED), 0);
sentReceiver = new BroadcastReceiver(){
@Override
public void onReceive(Context arg0, Intent arg1)
{ switch (getResultCode()) { case Activity.RESULT_OK:
    Toast.makeText(getApplicationContext(), "SMS sent",
Toast.LENGTH_SHORT).show();
    break;
    case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
    Toast.makeText(getApplicationContext(), "Generic failure", Toast.
LENGTH_SHORT).show();
    break;
    case SmsManager.RESULT_ERROR_NO_SERVICE:
    Toast.makeText(getApplicationContext(), "No service", Toast.
LENGTH_SHORT).show();
    break;
    case SmsManager.RESULT_ERROR_NULL_PDU:
    Toast.makeText(getApplicationContext(), "Null PDU", Toast.LENGTH_
SHORT).show();
    break;
    case SmsManager.RESULT_ERROR_RADIO_OFF:
    Toast.makeText(getApplicationContext(), "Radio off", Toast.LENGTH_
SHORT).show();
    break;
}
}
};

deliveredReceiver = new BroadcastReceiver(){ @Override
public void onReceive(Context arg0, Intent arg1)
{ switch (getResultCode()) { case Activity.RESULT_OK:
    Toast.makeText(getApplicationContext(), "SMS successfully delivered",
Toast.LENGTH_SHORT).show();
    break;
    case Activity.RESULT_CANCELED: Toast.makeText(getApplicationContext(),
    "Failure—SMS not delivered", Toast.LENGTH_SHORT).show();
    break;
}
}
};

registerReceiver(sentReceiver, new IntentFilter(SENT));
registerReceiver(deliveredReceiver, new IntentFilter(DELIVERED));

```

```

Button sendBtn = (Button) this.findViewById(R.id.send_button);
sendBtn.setOnClickListener(new Button.OnClickListener(){
    public void onClick(View v) {
        phoneNumber = (EditText) findViewById(R.id.recv_no);
        message = (EditText) findViewById(R.id.txt_msg);
        if(phoneNumber.getText().toString().trim().length() >0 && message.
        getText().toString().trim().length() >0) {
            SmsManager sms = SmsManager.getDefault();
            sms.sendTextMessage(phoneNumber.getText().toString(), null,
            message.getText().toString(), sentPendingIntent, delivered_pendint net);

        }
    else {
        Toast.makeText(SendSMSAppActivity.this, "Either phone number or
text is missing", Toast.LENGTH_SHORT).show();
    }
}
;
Button cancelBtn = (Button) this.findViewById(R.id.cancel_button);
cancelBtn.setOnClickListener(new Button.OnClickListener(){
    public void onClick(View v) { phoneNumber.setText(""); message.setText("");
    });
}
}
}

```

To send an SMS message with Java code, we use the `SmsManager` class. We cannot instantiate this class directly and must call the `getDefault()` static method to create its object. The method provided by the `SmsManager` class for sending SMS messages is the `sendTextMessage()` method.

The syntax for the `sendTextMessage()` method is:

**`sendTextMessage(recipient_phoneno, service_centeradd, sms_msg, sent_intent,`
`deliv-ery_intent)`**

where `recipient_phoneno` is the recipient's phone number, and `service_centeradd` is the Service center address. We use the value `null` for the default SMSC (Short Message Service Center). The `sms_msg` is the text message of the SMS, `sentIntent` is the pending Intent to invoke when the message is sent, and `delivery_Intent` is the pending

Intent to invoke when the message is delivered.

To monitor the status of the SMS message and confirm if it was sent correctly, we create two PendingIntent objects that are then passed as arguments to the sendTextMessage() method.

The two PendingIntent objects are created with the following statements:

```
String SENT = "SMS_SENT";
String DELIVERED = "SMS_DELIVERED";
final PendingIntent sentPendingIntent = PendingIntent.getBroadcast(this, 0, new
Intent(SENT), 0);
final PendingIntent delivered_pendingIntent = PendingIntent.getBroadcast(this, 0, new
Intent(DELIVERED), 0);
```

The two PendingIntent objects are passed into the last two arguments of the sendTextMessage() method:

```
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(phoneNumber.getText().toString(), null,
message.getText().toString(), sentPendingIntent, delivered_pendingIntent);
```

We are informed, via the two PendingIntent objects, whether the message was successfully sent, delivered, or failed. The SmsManager fires SENT and DELIVERED when the SMS messages are sent and delivered. The two PendingIntent objects are used to send broadcasts when an SMS message is sent or delivered. We also create and register two BroadcastReceivers, which listen for Intentsthat match SENT and DELIVERED as shown by the following statements:

```
registerReceiver(sentReceiver, new IntentFilter(SENT));
registerReceiver(deliveredReceiver, new IntentFilter(DELIVERED));
```

In the earlier section we saw how to define the BroadcastReceiver in the AndroidManifest.xmlfile, but here we are doing it dynamically. Within each BroadcastReceiver we override the onReceive() method and get the current result code via the getResultCode() method. Depending on the value of the result code, we display a successfully sent or failed message.

Note

On the Android emulator, only the sentPendingIntent PendingIntent object is fired, but not the deliveredPendingIntent PendingIntent object. On a real device, both PendingIntent objects fire.

To run the application and watch its output, we need to have two AVDs (Android Virtual Devices) running. From one AVD, we send the message. We use the other

AVD for receiving the message. We already have one AVD created called demoAVD, which we have been using to watch application results. We only need to create one more AVD. Select the Window, AVD Manager option to open the AVD Manager dialog box. Select New to create a new AVD. A dialog box opens, prompting for the name and other configuration settings. Assign the name testSMSAVD with the configuration settings shown in Figure.

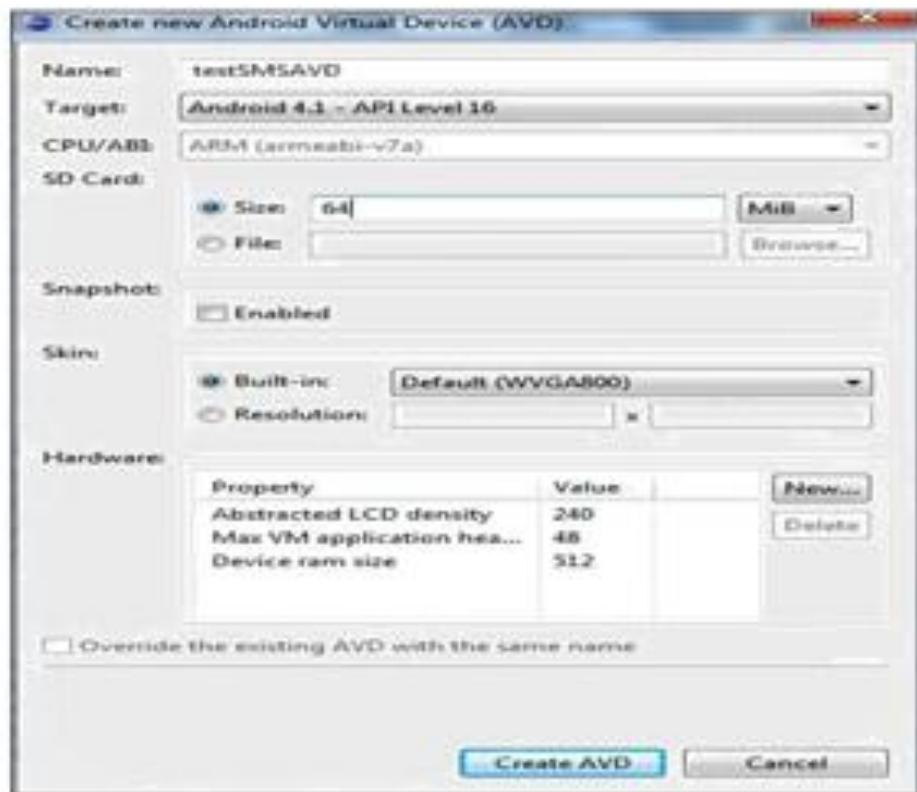


Figure. A dialog box showing configuration settings for the new AVD

Let's designate the SD Card size as 64MB. The Skin and Hardware attributes are set by default. Keeping the default values, click the Create AVD button to create the AVD.

Both the AVDs run in their respective windows. Android automatically assigns a unique number to each running AVD. For example, the first AVD has the title, 5554:demoAVD, and the other title displays 5556:testSMSAVD, as shown in Figure (left and right). The numbers 5554 and 5556 are the unique numbers assigned to each running emulator. The numbers keep incrementing with every additional emulator that is run.

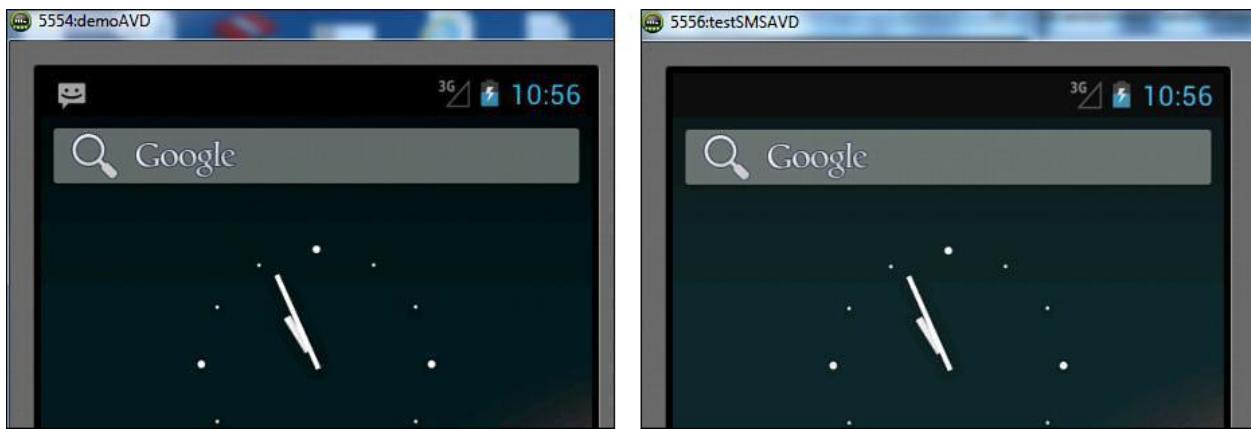


Figure (left) An AVD with the title 5554:demoAVD, and (right) an AVD with the title 5556:testSMSAVD.

Let's select the Run icon from the Eclipse toolbar (or press Ctrl+F11) to run our SMS application. We see the Android Device Chooser dialog box, asking us to select one of the running AVDs on which we want to run our SMS application. Let's select the 5554, that is, the demoAVD emulator, followed by the OK button. Our SMS application executes on the selected AVD, as shown in Figure (left). The title bar of the AVD 5554:demoAVD confirms that our SMS application is running on the selected emulator. We have applied validation checks on the two EditText controls that accept the phone number of the recipient and the SMS message. If either of the EditText controls is left empty, an error message, Either phone number or text is missing, is displayed. When we click the Send SMS button after supplying the recipient phone number and a text message, the SMS message is sent to the specified phone number, and a message, SMS sent, is displayed, as shown in Figure (middle). The SMS message is delivered to the 5556:testSMSAVD emulator, as shown in Figure (right).

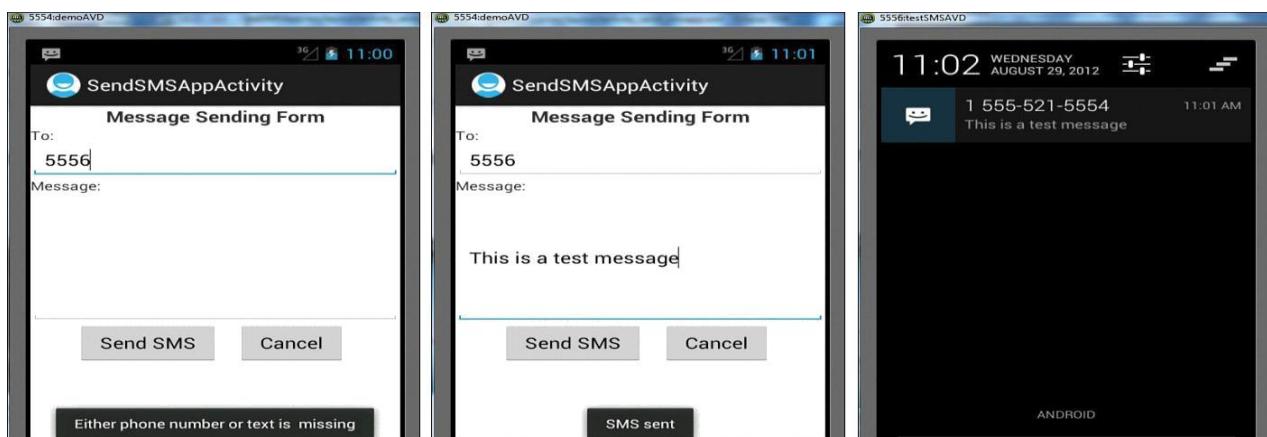


Figure. The error displayed when a field is missing, (left), an SMS message successfully delivered after delivery of the SMS message (middle), and notification of an SMS message being received (right)

7.4 RECEIVING SMS MESSAGES

To understand the concept of receiving SMS messages, let's create a new Android project called ReceiveSMSApp. When a device receives a new SMS message, a new broadcast Intent is fired with the android.provider.Telephony.SMS_RECEIVED action. To listen for this action, our application must register a BroadcastReceiver. That is, we need to create a receiver—a class that extends the BroadcastReceiver and then registers it in the manifest file. After we register the receiver, our application is notified whenever an SMS message is received. We see to the process of registering the receiver after creating it.

To create a receiver, add a class file in ReceiveSMSApp by right-clicking the src/com.androidunleashed.receivesmsapp folder in the Package Explorer window and select the New, Class option. Assign the name ReceiverSMS to the class file followed by clicking the Finish button. A Java file named ReceiverSMS.java is added to our project. To receive the incoming SMS message and display its details, write the code shown in Listing into the Java file ReceiverSMS.java.

Listing. Code Written into ReceiverSMS.java

```
package com.androidunleashed.receivesmsapp;

import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.Context;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class ReceiverSMS extends
    BroadcastReceiver { @Override
    public void onReceive(Context context, Intent intent)
    { Bundle bundle = intent.getExtras();
        SmsMessage[] msg = null;
        String str = "";
        if (bundle != null) {
            Object[] pdus = (Object[]) bundle.get("pdus");
            msg = new
```

```

SmsMessage[pdus.length];
for (int i=0; i<msg.length; i++){
    msg[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
    str += "SMS Received from: " + msg[i].getOriginatingAddress();
    str += ":";
    str += msg[i].getMessageBody().toString(); str += "\n";
}
Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
}
}
}

```

The broadcast Intent that is fired after receiving a new SMS message includes a bundle containing information about the received message. The information exists in the bundle in the form of an array of SMS PDUs. PDU stands for Protocol Data Unit and is used to encapsulate the SMS message. Hence, the PDU key is used to extract the array of SMS PDUs from the bundle. The array of SMS PDUs is thereafter converted into an array of SmsMessage objects. To convert each PDU byte array into an SMS Message object, the SmsMessage.createFromPdu() method is called, passing in each byte array. This code snippet shows how it is done:

```

Bundle bundle = intent.getExtras(); SmsMessage[] msg = null;
if (bundle != null) {
    Object[] pdus = (Object[]) bundle.get("pdus");
    msg = new SmsMessage[pdus.length];
    for (int i=0; i<msg.length; i++){
        msg[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
    }
}

```

We can see that the information of all received messages is gathered in the SmsMessage array msg. Each array element represents the complete information of a single received SMS that includes the originating address—the phone number, timestamp, and the message body. The following methods are used to fetch the SMS information from the SmsMessage array elements:

- **getOriginatingAddress()**—Fetches the phone number of the SMS recipient
- **getTimestampMillis()**—Fetches the time at which the SMS is received
- **getMessageBody()**—Fetches the message body

The code in Listing receives the SMS message and uses the getOriginatingAddress()

and getMessageBody() methods to fetch the phone number and message body of the received message. The phone number and message body are then displayed on the screen.

To listen for incoming messages, we need to register our ReceiverSMS Broadcast Receiver class using an Intent Filter that listens for the android.provider.Telephony.SMS_RECEIVED action String. To do so, the following code is added in the AndroidManifest.xml file:

```
<receiver android:name=".ReceiverSMS">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"
    /> </intent-filter>
</receiver>
```

Besides registering a receiver, our application has to seek permission to receive incoming SMS messages. To do this, add the android.permission.RECEIVE_SMS permission to the manifest file:

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

After we add the preceding code, the code in the AndroidManifest.xml file appears as shown in Listing.

Listing. Code Written into AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.receivesmsapp" android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity android:name=".ReceiveSMSAppActivity"
            android:label="@string/title_activity_receive_smsapp">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".ReceiverSMS">
            <intent-filter>
```

```

<action
    android:name="android.provider.Telephony.SMS_RECEIVED"/>
</intent-filter>
</receiver>
</application>
</manifest>

```

After running the application, we see the output shown in Figure

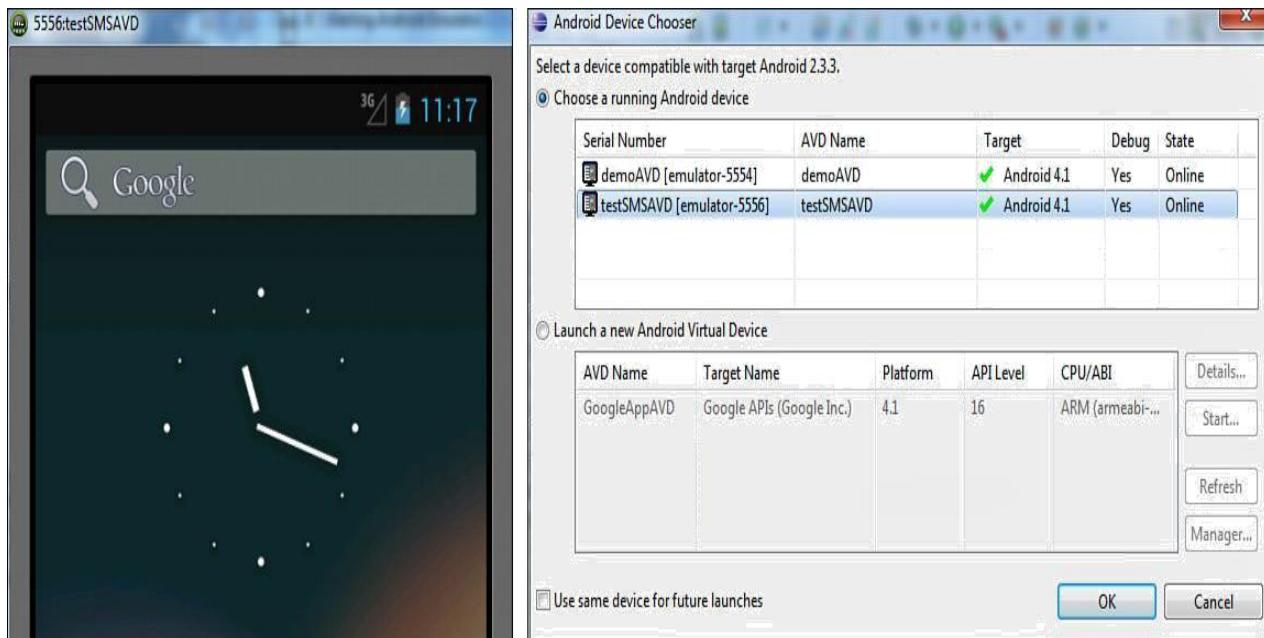


Figure. The output displayed on application startup

To see how an SMS message is received by this application, we need to first send a message using the SendSMSApp we just created. To run the SendSMSApp, let's run one more AVD. Select the Window, AVD Manager option. From the AVD Manager dialog box that pops up, select the testSMSAVD AVD and click the Start button. A dialog box showing Launch Options opens; click the Launch button to start the AVD with the default options.

The AVD runs, as shown in Figure (left). Close the AVD Manager window. Select SendSMSApp from the Project Explorer window and select the Run icon from the toolbar at the top. An Android Device Chooser dialog box opens, asking in which AVD we want to run the SentSMSApp, as shown in Figure (right).

The SendSMSApp application is launched in the 5556 AVD, as shown in Figure (left). In the Tobox, write the ID of the AVD through which our ReceiveSMSApp is running, that is, 5554. Write some text in the multiline message box and click the Send SMS button. The SMS message is sent from the application and received by the 5554 AVD, as is confirmed by the Toast message displaying the originating phone number and its message body (see Figure—middle). The SMS message is received and displayed in the 5554 AVD (see Figure—right).

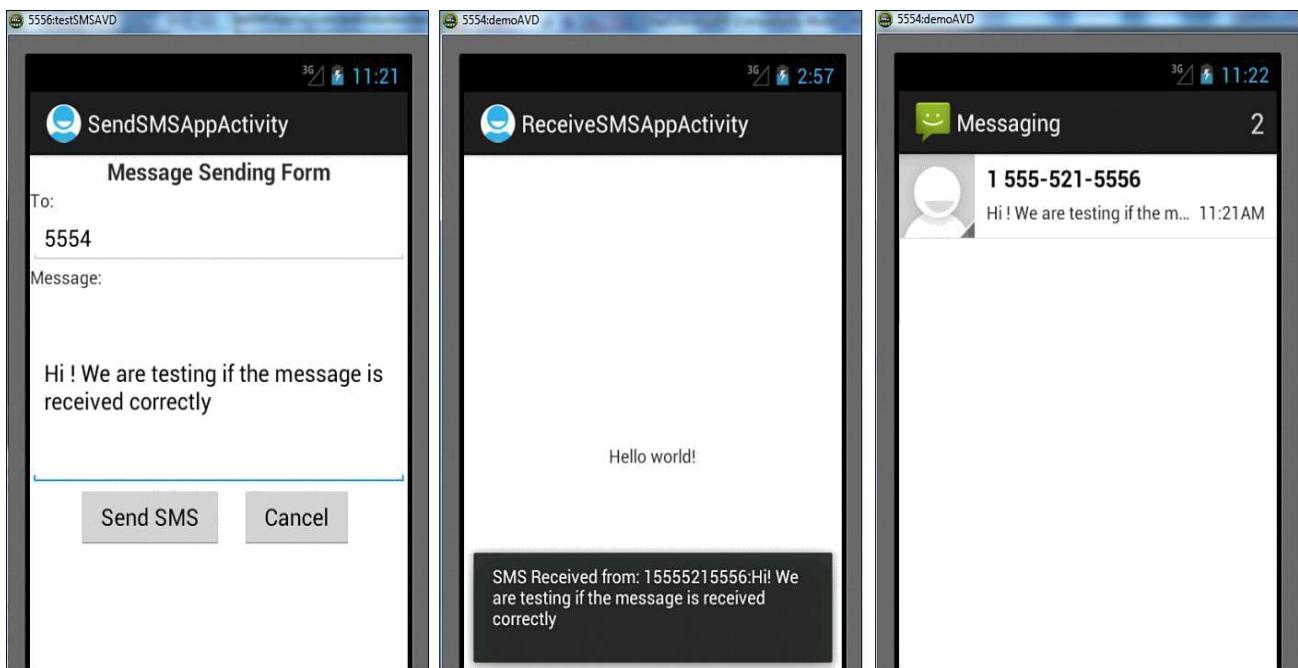


Figure. (left) Screen sending an SMS message, (middle) Toast displaying the sender information and received SMS body, and (right) the SMS message received by the receiver

7.5 SENDING EMAIL

To send an email with Android, we use the following Intent:

Intent.ACTION_SEND

The Intent.ACTION_SEND calls an existing email client to send an email. So, for sending email through the Android emulator, we need to first configure the email client. If the email client is not configured, the emulator does not respond to the Intent. Through the ACTION_SEND Intent, messages of different types, such as text or image, can be sent. The only thing we need to do is to set the type of the Intent

through the `setType()` method. For example, the following statement declares that the text data type will be sent through the Intent:

```
emailIntent.setType("plain/text");
```

The `emailIntent` is the `Intent.ACTION_SEND` object. After we launch this Intent, any application on the device that supports plain text messaging may handle this request. To let a user choose the email client to handle the Intent, we call `startActivity()` with the `createChooser()` method. As the name suggests, the `createChooser()` method prompts the user to choose the application to handle the Intent. This statement shows how to call `startActivity()` with the `createChooser()` method:

```
startActivity(Intent.createChooser(emailIntent, "Sending Email"));
```

This statement displays all the applications that are eligible to handle the Intent, allowing the user to choose the application to launch. If there is only one application able to handle the Intent, it is launched automatically without prompting the user.

To supply data for the email message fields, we set certain standard extras for the Intent. For example, we can set the following:

- **EXTRA_EMAIL**—Sets the To: address (email address of the receiver)
- **EXTRA_CC**—Sets the Cc: address (email address of the carbon copy receiver)
- **EXTRA_BCC**—Sets the Bcc: address (email address of the blind carbon copy receiver)
- **EXTRA_SUBJECT**—Sets the Subject of the email
- **EXTRA_TEXT**—Sets the body of the email

After setting the desired Intent's extras, launch the activity to initiate the sending email task.

Before we go ahead and create an application for sending email, let's configure the email client of the Android emulator in these steps:

- Start the emulator and then click on the Menu button.
- Click on the System settings option.
- From the Accounts section, click on the Add account button (see Figure—left). From the two options, Corporate and Email (see Figure—middle), click the Email option. We get the Account setup form where we need to enter our existing Email ID and password (see Figure—right) followed by clicking the Next button.

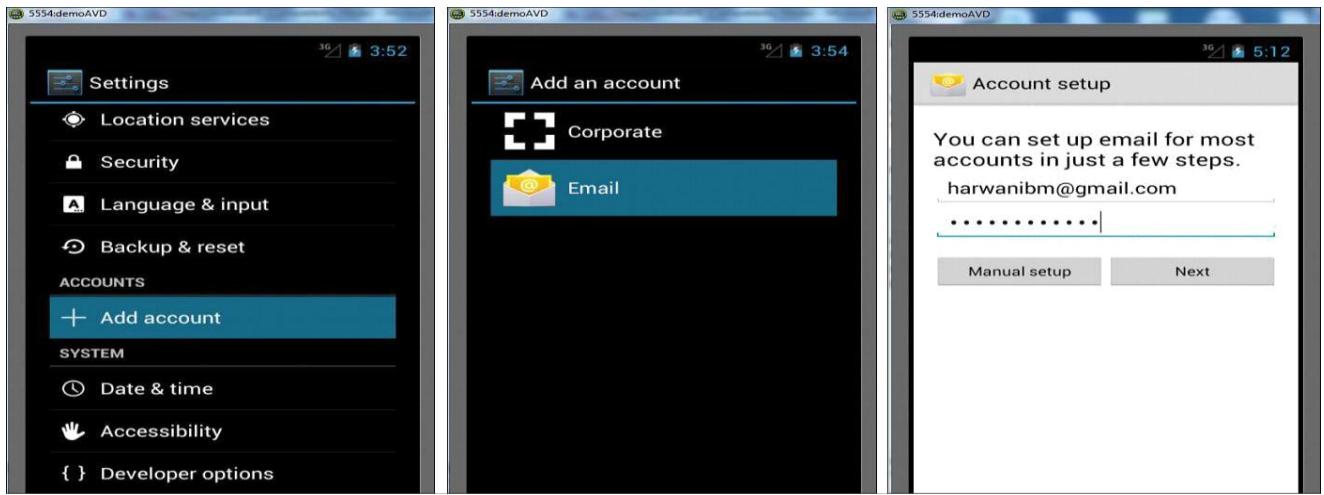


Figure. (left) Settings options in the Android emulator, (middle) two options of adding an account, and (right) account setup form to enter email ID and password

- The emulator checks for the incoming and outgoing servers and on finding them displays Account settings as shown in Figure (left). Select the required check boxes and then click the Nextbutton. Our email client is set up, and we are prompted to enter the name to be displayed on the outgoing messages (see Figure —right). Click the Next button to finish configuring the email client.

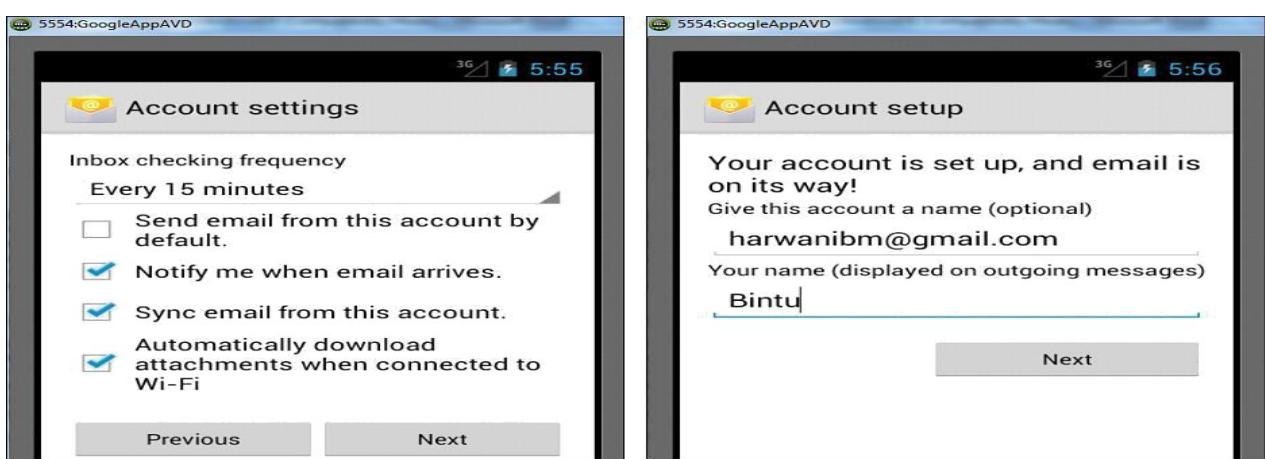


Figure. (left) The Account settings form, and (right) the Account setup form

Now the email client is successfully configured in the emulator, and we can go ahead and send mail. Let's create an application to send an email. Create a new Android project called SendEmailApp. In this application, we provide a user interface to enter

the To, Cc, and Bcc addresses of the email receivers; a Subject; and the body of the email. After we enter the required information, when the user clicks the Send button, the email client on the device is invoked. The email client page is auto-filled with the information entered through the application, and the email is sent after clicking the Send button from the email client.

To create the UI for entering email information, we need to use six TextView controls, five EditTextcontrols, and a Button control. To define these controls, write the code shown in Listing_nto the activity_send_email_app.xml layout file.

Listing. Code Written into activity_send_email_app.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent" >
    <TextView
        android:id="@+id/email_form"
        android:text = "Email Form" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:typeface="serif" android:textSize="18sp" android:textStyle="bold"
        android:padding="10dip" android:layout_centerHorizontal ="true"/>
    <TextView android:id="@+id/to_addressview" android:text = "To:"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_margin="10dip"
        android:layout_below="@+id/email_form" />
    <EditText android:id="@+id/toaddresses" android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:layout_below="@+id/email_form"
        android:layout_toRightOf="@+id/to_addressview" android:singleLine="true" />
    <TextView android:id="@+id/cc_addressview" android:text = "Cc:"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_below="@+id/to_addressview" android:layout_margin="10dip" />
    <EditText android:id="@+id/ccaddresses"
        android:layout_height="wrap_content" android:layout_width="match_parent"
        android:singleLine="true" android:layout_below="@+id/toaddresses"
        android:layout_toRightOf="@+id/cc_addressview" />
    <TextView android:id="@+id/bcc_addressview" android:text = "Bcc:"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_below="@+id/cc_addressview"
        android:layout_margin="10dip" />
```

```

<EditText android:id="@+id/bccaddresses" android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:singleLine="true"
    android:layout_below="@+id/ccaddresses"
    android:layout_toRightOf="@+id/bcc_addressview" />
<TextView android:id="@+id/subjectview" android:text = "Subject:"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_below="@+id/bcc_addressview" android:layout_margin="10dip"
    android:paddingTop="10dip"/>
<EditText android:id="@+id/emailsubject" android:layout_height="wrap_content"
    android:layout_width="match_parent" android:singleLine="true"
    android:layout_below="@+id/bccaddresses"
    android:layout_toRightOf="@+id/subjectview"
    android:layout_marginTop="10dip" />
<TextView android:id="@+id/emailtextview" android:text = "Message:"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_below="@+id/subjectview" android:layout_margin="10dip" />
<EditText android:id="@+id/emailtext"
    android:layout_height="wrap_content" android:layout_width="match_parent"
    android:lines="5" android:layout_below="@+id/emailsubject"
    android:layout_toRightOf="@+id/emailtextview" />
<Button android:id="@+id/send_button" android:text="Send"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_centerHorizontal="true" android:paddingLeft="25dip"
    android:paddingRight="25dip" android:layout_marginTop="10dip"
    android:layout_below="@+id/emailtext" />
</RelativeLayout>

```

We can see that the six TextView controls are set to display Email Form, To:, Cc:, Bcc:, Subject:, and Message. To identify these in the Java code, the five EditText controls are assigned the IDs toaddresses, ccaddresses, bccaddresses, emailsubject, and emailtext. The Button control is assigned the Send caption and is assigned the ID send_button. To read the addresses, subject, and email body entered by the user, and to use Intent.ACTION_SEND for sending email, write the code shown in Listing into the SendEmailAppActivity.java Java activity file.

Listing. Code Written into SendEmailAppActivity.java

```

package com.androidunleashed.sendemailapp;
import android.app.Activity;
import android.os.Bundle;

```

```

import android.view.View;
import android.content.Intent;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
public class SendEmailAppActivity extends Activity
{
    Button sendbutton;
    EditText toAddress, ccAddress, bccAddress, subject, emailMessage;
    String toAdds, ccAdds, bccAdds;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_send_email_app);
        sendbutton=(Button) findViewById(R.id.send_button);
        toAddress=(EditText) findViewById(R.id.toaddresses);
        ccAddress=(EditText) findViewById(R.id.ccaddresses);
        bccAddress=(EditText) findViewById(R.id.bccaddresses);
        subject=(EditText) findViewById(R.id.emailsubject);
        emailMessage=(EditText) findViewById(R.id.emailtext);
        sendbutton.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                final Intent emailIntent = new Intent(Intent.ACTION_SEND);
                if(toAddress.getText().length() >0) {
                    toAdds = ""+toAddress.getText().toString()+"";
                    emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]{ toAdds });
                }
                if(ccAddress.getText().length() >0) {
                    ccAdds = ""+ccAddress.getText().toString()+"";
                    emailIntent.putExtra(Intent.EXTRA_CC, new String[]{ ccAdds });
                }
                if(bccAddress.getText().length() >0) {

```

```

        bccAdds = ""+bccAddress.getText().toString()+"";
        emailIntent.putExtra(Intent.EXTRA_BCC, new String[]{ bccAdds});
    }
    emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject.getText().
toString());
    emailIntent.putExtra(Intent.EXTRA_TEXT, emailMessage.getText());
    emailIntent.setType("plain/text");
    startActivity(Intent.createChooser(emailIntent, "Sending Email"));
}
}); } }

```

Here, the toaddresses, ccaddresses, bccaddresses, emailsubject, and emailtext EditText controls are accessed and mapped to their respective EditText objects. Similarly, the Button control with the send_button ID is accessed and assigned to the sendbutton Buttonobject. The ClickListener event listener is associated with the Button control. After we click the Button, the onClick() callback method is invoked. In the onClick() method,

an Intent object called emailIntent is created, with its action set to Intent.ACTION_SEND. Also, to auto-fill the email client's fields, the Intent extras, EXTRA_EMAIL, EXTRA_CC, EXTRA_BCC, EXTRA SUBJECT, and EXTRA_TEXT, are initialized with the data entered by the user in the EditText controls. Finally, the email is sent by launching the email client installed on the device.

After running the application, we see a user interface to enter addresses of the receiver(s), subject, and email body, as shown in Figure _ (top left). When the user clicks the Send button, the email client is invoked. The fields of the email client are automatically filled with data entered in the EditTextcontrols, as shown in Figure (top right). The email is sent to the recipient after it pipes through the email client, as shown in Figure (bottom).

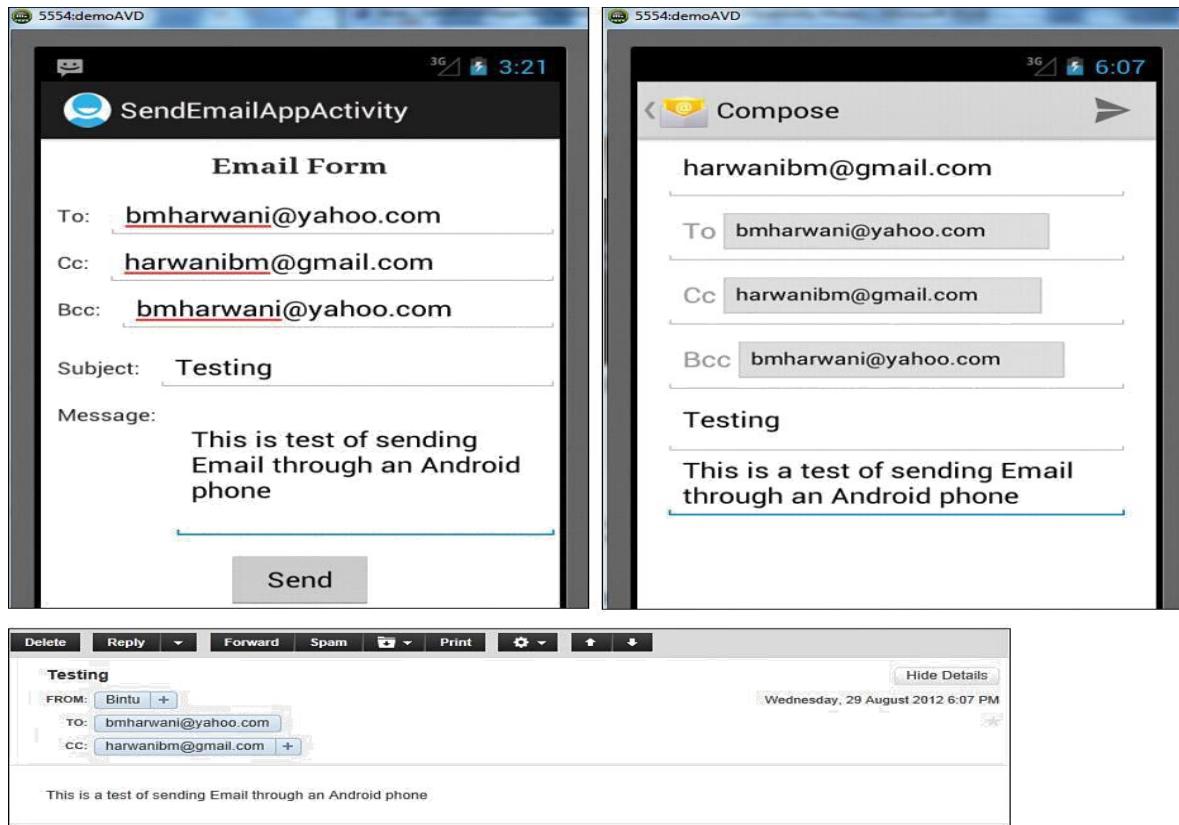


Figure (top left) Information entered into the Email Form, (top right) information entered into the Email Form appears in the email client fields, and (bottom) email received by the recipient

7.6 WORKING WITH THE TELEPHONY MANAGER

The Android telephony APIs include the Telephony Manager that accesses the telephony services on the device and enables us to

- \{Provide a user interface for entering or modifying the phone number to dial.
- \{Implement call handling in the application.
- \{Register and monitor telephony state changes.
- \{Get subscriber information.

Making the Outgoing Call

The simplest way to make an outgoing call is to invoke the Dialer application by using the Intent.ACTION_CALL action. The sample code for doing so follows:

```
Intent callIntent = new
Intent(Intent.ACTION_CALL);
```

```
callIntent.setData(Uri.parse("tel:1111122222"));
startActivity(callIntent);
```

The preceding code initiates a phone call to 1111122222 using the system in-call Activity. For making the phone call, the application must have the permission to invoke the Dialer application. That is, to use this action, the application must request the CALL_PHONE uses- permission by adding the following statement to the manifest file:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Listening for Phone State Changes

- . The phone state is represented by the constants shown here:
 - **CALL_STATE_IDLE**—The phone is in an idle state.
 - **CALL_STATE_RINGING**—A phone call has arrived.
 - **CALL_STATE_OFFHOOK**—The phone is off-hook.

To access the phone state information, the application must have permission for doing so. To ask permission, add following statement to the manifest file:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Let's see how to make a phone call and listen to the phone state. Create a new Android project called PhoneCallApp. In this application, there are two controls: Button and TextView. The Buttoncontrol places the phone call, and the TextView displays the phone state.

Listing 11.17. Code Written into activity_phone_call_app.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical">
    <Button android:id="@+id/callbutton"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="Make Phone Call" android:layout_gravity="center" />
    <TextView android:id="@+id/messageview"
        android:layout_width="wrap_content" android:layout_height="match_parent"
        android:layout_gravity="center" />
</LinearLayout>
```

We can see that the Button control is assigned the caption Make Phone Call and

assigned the ID callbutton. The TextView is assigned the ID messageview. The IDs are used to access these controls in the Java code.

To make the outgoing call and listen to the phone state changes, write the code shown in Listing into PhoneCallAppActivity.java.

Listing. Code Written into PhoneCallAppActivity.java

```
package com.androidunleashed.phonecallapp;

import android.app.Activity;
import android.os.Bundle;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.view.View;
import android.widget.TextView;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.util.Log;

public class PhoneCallAppActivity extends Activity
    { @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_phone_call_app);
            MyPhoneCallListener phoneListener = new MyPhoneCallListener();
            TelephonyManager telephonyManager = (TelephonyManager)
                getSystemService(Context.TELEPHONY_SERVICE);
            telephonyManager.listen(phoneListener,PhoneStateListener.LISTEN_CALL_STATE);
            Button callButton = (Button) findViewById(R.id.callbutton);
            callButton.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View arg0) {
                    Intent callIntent = new Intent(Intent.ACTION_CALL);
                    callIntent.setData(Uri.parse("tel:1111122222"));
                    startActivity(callIntent);
                }
            });
        }
    }
```

```

}

class MyPhoneCallListener extends PhoneStateListener {
    TextView messageview = (TextView)findViewById(R.id.messageview);
    String msg;

    @Override
    public void onCallStateChanged(int state, String incomingNumber)
    { super.onCallStateChanged(state, incomingNumber);
        switch(state){
            case TelephonyManager.CALL_STATE_IDLE:
                msg= "Call state is idle";
                Log.d("idle", msg);
                break;
            case TelephonyManager.CALL_STATE_RINGING:
                msg = "Call state is Ringing. Number is "+ incomingNumber;
                Log.d("ringing", msg);
                break;
            case TelephonyManager.CALL_STATE_OFFHOOK: msg = "Call state is
                OFFHOOK";
                Log.d("offhook", msg);
                break;
            default:
                msg = "Call state is" + state + ". Number is " + incomingNumber;
                Log.d("state", msg);
                break;
        }
        messageview.setText(msg);
    }
}
}
}

```

We can see that to make an outgoing call, an Intent object called callIntent is created with an Intent.ACTION_CALL action. The phone number to call is set to 1111122222. The PhoneStateListener class is implemented to listen for the phone state changes. When the phone state changes, the onCallStateChanged() method is called, which displays the phone state through the TextView control.

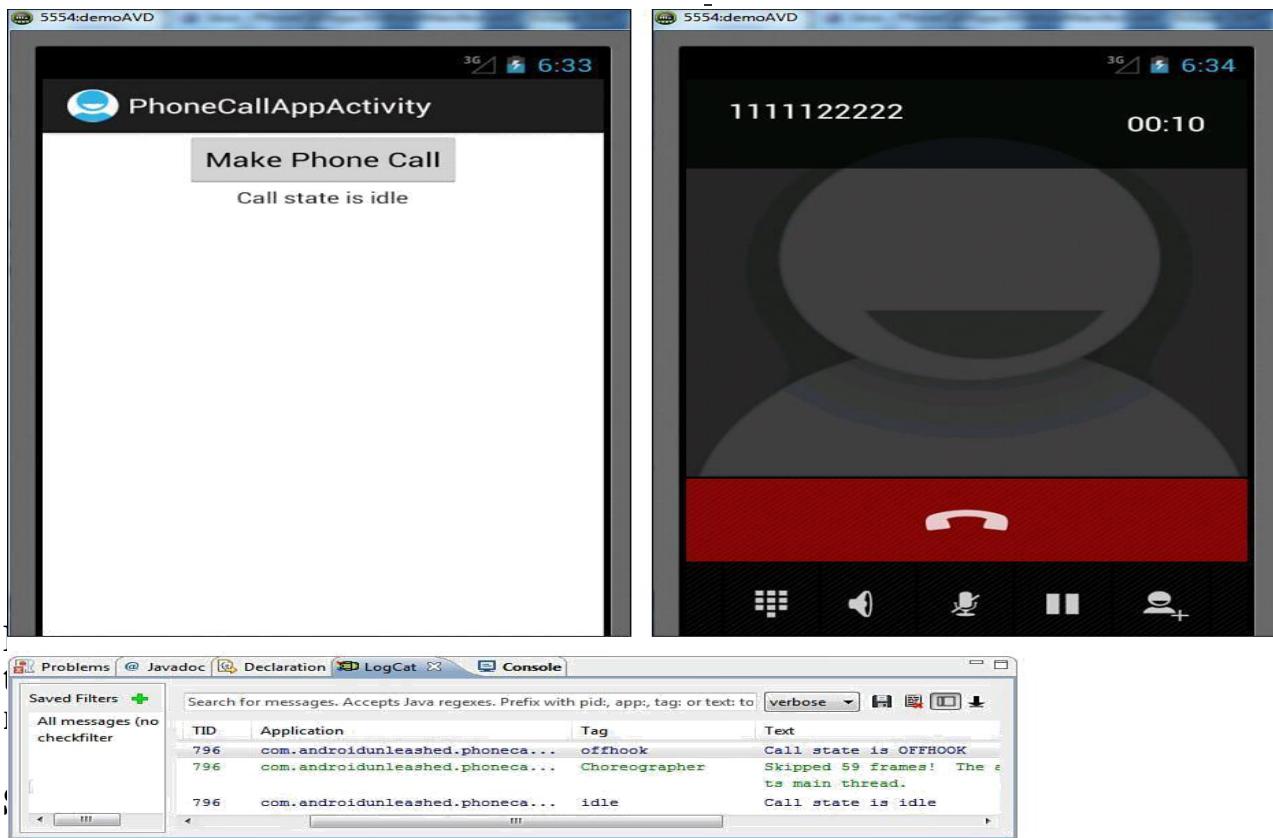
Before running the application, we need to add two permissions to the application. The first permission is for making phone calls, and the second is for accessing the phone state information. After we add the two permissions, AndroidManifest.xml

appears as shown in Listing. Only the code in bold is added; the rest is the default code.

Listing. Code in AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidunleashed.phonecallapp" android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />
    <b><uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" /></b>
    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity android:name=".PhoneCallAppActivity"
            android:label="@string/title_activity_phone_call_app"
            > <intent-filter>
                <action android:name="android.intent.action.MAIN" /> <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

After running the application, we see the Button and TextView controls on the screen. Because no phone call has been made, the TextView shows the text Call state is idle (see Figure — top left). After we click the Make Phone Call button, the phone call is made to the phone number specified in the activity file, as shown in Figure (top right). The log messages showing the phone state are displayed in Figure (bottom).



In this chapter, we learned about broadcast receivers. We saw how to broadcast and receive the broadcasted Intent. We saw how the notification system is used, and how the notification is created, configured, and displayed in the status bar. We saw the procedure for sending and receiving SMS messages using Java code. Finally, we saw how to send email and use the Telephony Manager in making phone calls.

In the next chapter, we learn how to define, create, use, and register content providers. We also learn how to define a database, content URI, and MIME types. Also we learn to implement the `getType`, `query`, `insert`, `update`, and `delete` methods required to make a content provider functional.

CHAPTER 7

PRACTICAL PROGRAMS

Ex.No: 1

Date : _____

Develop an application that uses GUI Components, Fonts and Colors

AIM:

To develop an application that uses GUI Components, Fonts and Colors.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_1.
3. Go to package explorer in the left hand side. Select the project Ex_No_1.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. One TextView with text MAD Lab
 - b. Three Buttons with labeled as Change Font Size, Change Font Color and Change Font Style
7. Again go to package explorer in the left hand side. Select the project Ex_No_1.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as, actions of buttons.
10. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    tools:context="com.example.ex_no_1.MainActivity" >  
    <TextView  
        android:id="@+id/textView1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignParentTop="true"
```

```

        android:layout_centerHorizontal="true"
        android:layout_marginTop="53dp"
        android:text="MAD Lab"
        android:textAppearance="?android:attr/textAppearanceLarge"
        tools:ignore="HardcodedText" />
<Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="64dp"
        android:text="Change Font Size"
        tools:ignore="HardcodedText" />
<Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/button1"
        android:text="Change Font Color"
        tools:ignore="HardcodedText" />
<Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/button2"
        android:text="Change Font Style"
        tools:ignore="HardcodedText" />
</RelativeLayout>
```

MainActivity.java:

```

package com.example.ex_no_1;
import android.support.v7.app.ActionBarActivity;
import android.graphics.Color;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
```

```
public class MainActivity extends ActionBarActivity {  
    float font = 20;  
    int count = 1;  
    Button b1,b2,b3;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        final TextView t1 = (TextView) findViewById(R.id.textView1);  
        t1.setTextSize(15);  
        b1 = (Button) findViewById(R.id.button1);  
        b1.setOnClickListener(new OnClickListener() {  
            public void onClick(View view) {  
                t1.setTextSize(font);  
                font = font + 5;  
                if (font == 50)  
                    font = 20;  
            }  
        });  
        b2 = (Button) findViewById(R.id.button2);  
        b2.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View view) {  
                switch (count) {  
                    case 1:  
                        t1.setTextColor(Color.parseColor("#7f00ff"));  
                        break;  
                    case 2:  
                        t1.setTextColor(Color.parseColor("#00FF00"));  
                        break;  
                    case 3:  
                        t1.setTextColor(Color.parseColor("#FF0000"));  
                        break;  
                    case 4:  
                        t1.setTextColor(Color.parseColor("#0000FF"));  
                        break;  
                }  
                count++;  
                if (count == 5)  
                    count = 1;  
            }  
        });  
        b3 = (Button) findViewById(R.id.button3);  
        b3.setOnClickListener(new OnClickListener() {  
            @Override  
            public void onClick(View view) {  
                switch (count) {
```

```

case 1:
t1.setTypeface(Typeface.DEFAULT, Typeface.ITALIC);
break;
case 2:
t1.setTypeface(Typeface.MONOSPACE, Typeface.NORMAL);
break;
case 3:
t1.setTypeface(Typeface.SANS_SERIF, Typeface.BOLD);
break;
case 4:
t1.setTypeface(Typeface.SERIF, Typeface.BOLD_ITALIC);
break;
}c
ount++;
if (count == 5)
count = 1;
}
});
}
}
}

```

OUTPUT:



RESULT:

Thus the application that uses GUI Components, Fonts and Colors has been developed and the output was verified.

Ex. No : 2	Develop an application that uses Layout Managers and Event Listeners
Date :	

AIM:

To develop an application that uses Layout Managers and Event Listeners.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_2.
3. Go to package explorer in the left hand side. Select the project Ex_No_2.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. Four TextViews with texts as Name, Gender, Degree and Programming Knowledge
 - b. One EditText
 - c. One Spinner
 - d. One RadioGroup with two RadioButtons labeled as B.E. CSE and B.Tech. IT
 - e. One RatingBar
 - f. One Button with labeled as SUBMIT
7. Again go to package explorer in the left hand side. Select the project Ex_No_2.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as, actions of button.
10. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_2.MainActivity" >
```

```
<TextView
```

```

        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Name"

        android:textAppearance="?android:attr/textAppearanceMedium"
        tools:ignore="HardcodedText" />

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/textView1"
    android:layout_marginLeft="14dp"
    android:layout_toRightOf="@+id/textView1"
    android:ems="10" tools:ignore="TextFields" >
    <requestFocus />
</EditText>

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/editText1"
    android:layout_marginTop="14dp" android:text="Gender"

    android:textAppearance="?android:attr/textAppearanceMedium"
    tools:ignore="HardcodedText" />

<Spinner
    android:id="@+id/spinner1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_alignTop="@+id/textView2"
    android:entries="@array/Gender" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"

```

```
    android:layout_below="@+id/spinner1"
    android:text="Degree"

    android:textAppearance="?android:attr/textAppearanceMedium"
    tools:ignore="HardcodedText" />

<RadioGroup
    android:id="@+id/radioGroup1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/spinner1"
    android:layout_below="@+id/spinner1" >

    <RadioButton
        android:id="@+id/radio0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true" android:text="B.E. CSE"
        tools:ignore="HardcodedText" />

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B.Tech IT"
        tools:ignore="HardcodedText" />

</RadioGroup>

<RatingBar
    android:id="@+id/ratingBar1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView4"
    android:layout_below="@+id/textView4" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView3"
    android:layout_below="@+id/radioGroup1"
    android:text="Programming Knowledge"
```

```

        android:textAppearance="?android:attr/textAppearanceMedium"
        tools:ignore="HardcodedText" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/ratingBar1"
        android:layout_centerHorizontal="true"
        android:text="SUBMIT" tools:ignore="HardcodedText" />

</RelativeLayout>

```

MainActivity.java:

```

package com.example.ex_no_2;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;

import android.view.View;

import android.view.View.OnClickListener;
import android.widget.Button;

import android.widget.EditText;

import android.widget.RadioButton;

import android.widget.RadioGroup;

import android.widget.RadioGroup.OnCheckedChangeListener;
import android.widget.RatingBar;

import android.widget.RatingBar.OnRatingBarChangeListener;
import android.widget.Spinner;

import android.widget.Toast;

public class MainActivity extends ActionBarActivity {
    String name,gender,dept;

    float prog;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final EditText e=(EditText)findViewById(R.id.editText1);

```

```

RadioGroup rg=(RadioGroup)findViewById(R.id.radioGroup1);
final RadioButton r1=(RadioButton)findViewById(R.id.radio0);
final RadioButton r2=(RadioButton)findViewById(R.id.radio1);
final Spinner s=(Spinner)findViewById(R.id.spinner1);
RatingBar rb=(RatingBar)findViewById(R.id.ratingBar1);
Button b=(Button)findViewById(R.id.button1);
rg.setOnCheckedChangeListener(
    new OnCheckedChangeListener()
    {
        @Override
        public void onCheckedChanged(RadioGroup arg0, int arg1) {

            // TODO Auto-generated method stub
            if(r1.isChecked()==true)
                dept="B.E. CSE";
            if(r2.isChecked()==true)
                dept="B.Tech IT";
        }
    });
rb.setOnRatingBarChangeListener(
    new OnRatingBarChangeListener()
    {
        @Override
        public void onRatingChanged(RatingBar arg0, float arg1,
                                    boolean arg2) {

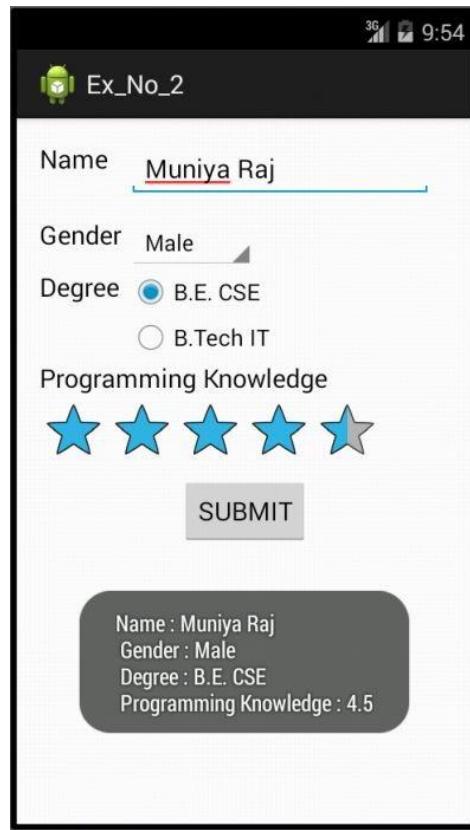
            // TODO Auto-generated method stub

            prog=arg1;
        }
    });
b.setOnClickListener(
    new OnClickListener()
    {
        @Override
        public void onClick(View arg0) {

            // TODO Auto-generated method stub
            name=e.getText().toString();
            gender=s.getSelectedItem().toString();
            Toast.makeText(getApplicationContext(), "Name : "+name+"\n Gender : "+gender+"\n Degree : "+dept+"\n Programming Knowledge : "+prog, Toast.LENGTH_LONG).show();
        }
    });
}
}

```

OUTPUT:



RESULT:

Thus the application that uses Layout Managers and Event Listener has been developed and the output was verified.

Ex. No : 3	
Date :	

Develop a native calculator application

AIM:

To develop a native calculator application.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_3.
3. Go to package explorer in the left hand side. Select the project Ex_No_3.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. Two EditTexts with hints Enter the first number and Enter the second number
 - b. Four Buttons with labeled as ADD, SUB, MUL and DIV
7. Again go to package explorer in the left hand side. Select the project Ex_No_3.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as, actions of button.
10. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_3.MainActivity" >

    <EditText android:id="@+id/editText1"
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true" android:ems="10"

        android:hint="Enter the first number"
```

```
tools:ignore="TextFields,HardcodedText" >

    <requestFocus />
</EditText>

<EditText
    android:id="@+id/editText2"

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/editText1" android:ems="10"

    android:hint="Enter the second number"
    tools:ignore="TextFields,HardcodedText" />

<Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/button3" android:text="DIV"
    tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/editText2" android:text="ADD"
    tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/button1" android:text="SUB"
    tools:ignore="HardcodedText" />

<Button
```

```

        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/button2" android:text="MUL"
        tools:ignore="HardcodedText" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button4"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="22dp" android:text=""

    android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>

```

MainActivity.java:

```

package com.example.ex_no_3;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;

import android.view.View;

import android.view.View.OnClickListener;
import android.widget.Button;

import android.widget.EditText; import
android.widget.TextView;

public class MainActivity extends ActionBarActivity {

    int n1,n2;

    float num1,num2;
    @Override

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final EditText e1=(EditText)findViewById(R.id.editText1); final
        EditText e2=(EditText)findViewById(R.id.editText2);   Button
        b1=(Button)findViewById(R.id.button1);
    }
}

```

```

        Button b2=(Button)findViewById(R.id.button2);  Button
        b3=(Button)findViewById(R.id.button3);           Button
        b4=(Button)findViewById(R.id.button4);
        FinalTextviewt=(TextView)findViewById(R.id.textView1);
        b1.setOnClickListener(
            new OnClickListener()
            {
                @Override
                public void onClick(View arg0) {

                    // TODO Auto-generated method stub
                    n1=Integer.parseInt(e1.getText().toString());
                    n2=Integer.parseInt(e2.getText().toString());
                    t.setText(e1.getText().toString()+
                    "+e2.getText().toString()+" = "+(n1+n2));
                }
            });
        b2.setOnClickListener(
            new OnClickListener()
            {
                @Override
                public void onClick(View arg0) {

                    // TODO Auto-generated method stub
                    n1=Integer.parseInt(e1.getText().toString());
                    n2=Integer.parseInt(e2.getText().toString());
                    t.setText(e1.getText().toString()+
                    "+e2.getText().toString()+" = "+(n1-n2));
                }
            });
        b3.setOnClickListener(
            new OnClickListener()
            {
                @Override
                public void onClick(View arg0) {

                    // TODO Auto-generated method stub
                    n1=Integer.parseInt(e1.getText().toString());
                    n2=Integer.parseInt(e2.getText().toString());

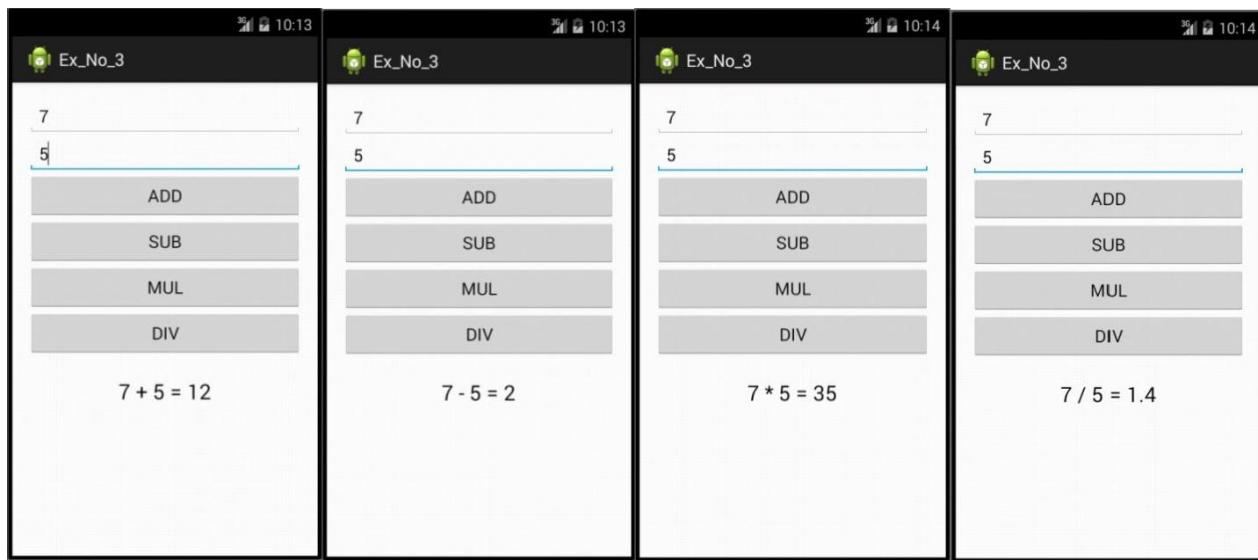
                    t.setText(e1.getText().toString()+
                    "+e2.getText().toString()+" = "+(num1/num2));
                }
            });
        b4.setOnClickListener(
            new OnClickListener()
            {
                @Override
                public void onClick(View arg0) {

                    // TODO Auto-generated method stub
                    num1=Float.parseFloat(e1.getText().toString());
                    num2=Float.parseFloat(e2.getText().toString());
                    t.setText(e1.getText().toString()+
                    "+e2.getText().toString()+" = "+(num1/num2));
                }
            });
    }
}

```

```
    });
}
}
```

OUTPUT:



RESULT:

Thus the native calculator application has been developed and the output was verified.

AIM:

To develop an application that draws basic graphical primitives on the screen.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_7.
3. Go to package explorer in the left hand side. Select the project Ex_No_7.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop only one ImageView
7. Again go to package explorer in the left hand side. Select the project Ex_No_6.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as drawing the graphical primitives.
10. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent" android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    tools:context="com.example.ex_no_7.MainActivity" >  
  
    <ImageView  
        android:id="@+id/imageView1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignParentBottom="true"  
        android:layout_alignParentLeft="true"  
        android:layout_alignParentRight="true"  
        android:layout_alignParentTop="true"  
        android:src="@drawable/ic_launcher"  
        tools:ignore="ContentDescription" />  
  
</RelativeLayout>
```

MainActivity.java:

```
package com.example.ex_no_7;  
  
import android.support.v7.app.ActionBarActivity;  
import android.annotation.SuppressLint;
```

```

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.Display;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.ImageView;
@SuppressWarnings("ClickableViewAccessibility")
public class MainActivity extends ActionBarActivity implements OnTouchListener { ImageView iv;

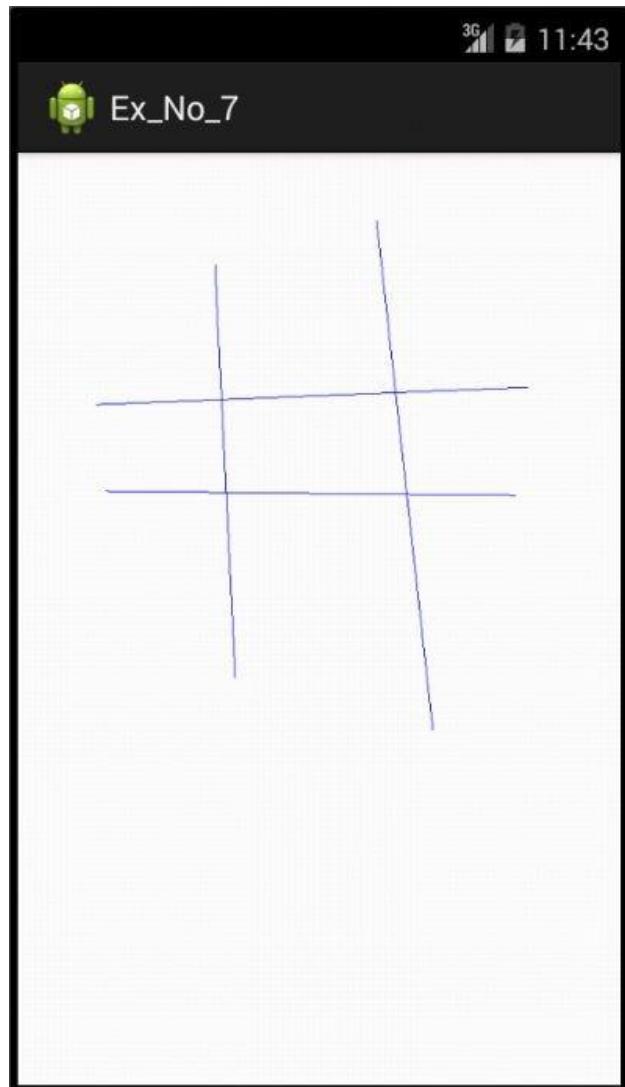
    Bitmap b; Canvas c; Paint p;
    float dx=0,dy=0,ux=0,uy=0;
    @SuppressWarnings("deprecation")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        iv=(ImageView)this.findViewById(R.id.imageView1);
        Display d = getWindowManager().getDefaultDisplay();
        float dw = d.getWidth();
        float dh = d.getHeight();
        b = Bitmap.createBitmap((int) dw, (int) dh,Bitmap.Config.ARGB_8888); c = new Canvas(b);
        p = new Paint();
        p.setColor(Color.BLUE);
        iv.setImageBitmap(b);
        iv.setOnTouchListener(this);
    }
    @Override
    public boolean onTouch(View v, MotionEvent event) {

        // TODO Auto-generated method stub
        int action = event.getAction();
        switch (action)
        {
            case MotionEvent.ACTION_DOWN:
                dx = event.getX();
                dy = event.getY();
                break;
            case MotionEvent.ACTION_MOVE:
                break;
            case MotionEvent.ACTION_UP:
                ux = event.getX();
                uy = event.getY();
                c.drawLine(dx, dy, ux, uy, p);
                iv.invalidate();
                break;
            case MotionEvent.ACTION_CANCEL:
                break;
            default:
                break;
        }

        return true;
    }
}

```

OUTPUT:



RESULT:

Thus the application that draws basic graphical primitives on the screen has been developed and the output was verified.

AIM:

To develop an application that makes use of database.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_4.
3. Go to package explorer in the left hand side. Select the project Ex_No_4.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. Three TextViews with texts as Reg.No., Name and Marks
 - b. Three EditTexts
 - c. Five Buttons with labeled as ADD, VIEW, VIEW ALL, UPDATE and DELETE
7. Again go to package explorer in the left hand side. Select the project Ex_No_4.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as, actions of button.
10. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_5.MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Reg. No."
        android:textAppearance="?android:attr/textAppearanceMedium"
        tools:ignore="HardcodedText" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/textView1"
        android:layout_toRightOf="@+id/textView1"
        android:ems="10" android:inputType="number" >
```

```

</EditText>

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/editText1"
    android:layout_marginTop="20dp"
    android:text="Name"
    android:textAppearance="?android:attr/textAppearanceMedium"
    tools:ignore="HardcodedText" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_below="@+id/editText2"
    android:layout_marginTop="26dp"
    android:text="Marks"
    android:textAppearance="?android:attr/textAppearanceMedium"
    tools:ignore="HardcodedText" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/textView3"
    android:layout_alignLeft="@+id/editText2"
    android:ems="10"
    android:inputType="number" />

<EditText
    android:id="@+id/editText2" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_alignLeft="@+id/editText1"
    android:ems="10"
    tools:ignore="TextFields" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textView3"
    android:layout_marginTop="32dp"
    android:text="ADD"
    tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button3" android:layout_width="wrap_content"
    android:layout_height="wrap_content"

```

```

        android:layout_alignBaseline="@+id/button2"
        android:layout_alignBottom="@+id/button2"
        android:layout_alignParentRight="true"
        android:text="VIEW ALL"
        tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button1"
    android:layout_alignBottom="@+id/button1"
    android:layout_alignLeft="@+id/editText3"
    android:layout_marginLeft="24dp"
    android:text="VIEW" tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button1"
    android:layout_below="@+id/button1"
    android:layout_marginLeft="27dp"
    android:layout_marginTop="18dp"
    android:text="UPDATE"
    tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button4"
    android:layout_alignBottom="@+id/button4"
    android:layout_marginLeft="20dp"
    android:layout_toRightOf="@+id/button4"
    android:text="DELETE"
    tools:ignore="HardcodedText" />

</RelativeLayout>

```

MainActivity.java:

```

package com.example.ex_no_4;

import android.support.v7.app.ActionBarActivity;
import android.app.AlertDialog.Builder;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
public class MainActivity extends ActionBarActivity {

```

```

    EditText name,regno,mark;

```

```

Button btnAdd,btnDelete,btnUpdate,btnView,btnViewAll;
SQLiteDatabase db;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    regno= (EditText)findViewById(R.id.editText1);
    name= (EditText)findViewById(R.id.editText2);
    mark=(EditText)findViewById(R.id.editText3);
    btnAdd=(Button)findViewById(R.id.button1);
    btnView=(Button)findViewById(R.id.button2);
    btnViewAll=(Button)findViewById(R.id.button3);
    btnUpdate=(Button)findViewById(R.id.button4);
    btnDelete=(Button)findViewById(R.id.button5);
    db=openOrCreateDatabase("Students", Context.MODE_PRIVATE, null);
    db.execSQL("CREATE TABLE IF NOT EXISTS student(regno VARCHAR,name VARCHAR,mark
    VARCHAR');");
    btnAdd.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View arg0) {
            // TODO Auto-generated method stub
            if(regno.getText().toString().trim().length()==0||name.getText().toString().trim().length()==0|
            |mark.getText().toString().trim().length()==0)
            {
                showMessage("Error", "Please enter all values");
                return;
            }
            db.execSQL("INSERT INTO student
VALUES('"+regno.getText()+"','"+name.getText()+"','"+mark.getText()+"');");
            showMessage("Success", "Record added");
            clearText();
        }
    });
    btnDelete.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            if(regno.getText().toString().trim().length()==0)
            {
                showMessage("Error", "Please enter Reg. No.");
                return;
            }
            Cursor c=db.rawQuery("SELECT * FROM student WHERE
regno='"+regno.getText()+"'", null);
            if(c.moveToFirst())
            {
                db.execSQL("DELETE FROM student WHERE regno='"+regno.getText()+"'");
                showMessage("Success", "Record Deleted");
            }
            else
            {
                showMessage("Error", "Invalid Reg. No.");
            }
        }
    });
}

```

```

        clearT      ext();
    }
    btnUpdate.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            if(regno.getText().toString().trim().length()==0)
            {
                showMessage("Error", "Please enter Reg. No.");
                return;
            }
            Cursor c=db.rawQuery("SELECT * FROM student WHERE
regno='"+regno.getText()+"'", null);
            if(c.moveToFirst())
            {
                db.execSQL("UPDATE student SET
name='"+name.getText()+"',mark='"+mark.getText()+"' WHERE regno='"+regno.getText()+"'");
                showMessage("Success", "Record Modified");
            }
            else
            {
                showMessage("Error", "Invalid Reg. No.");
                clearText();
            }
        });
        btnView.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                if(regno.getText().toString().trim().length()==0)
                {
                    showMessage("Error", "Please enter Reg. No.");
                    return;
                }
                Cursor c=db.rawQuery("SELECT * FROM student WHERE
regno='"+regno.getText()+"'", null);
                if(c.moveToFirst())
                {
                    name.setText(c.getString(1));
                    mark.setText(c.getString(2));
                }
                else
                {
                    showMessage("Error", "Invalid Reg. No.");
                    clearText();
                }
            });

```

```

btnViewAll.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v) {

        // TODO Auto-generated method stub
        Cursor c=db.rawQuery("SELECT * FROM student", null);
        if(c.getCount()==0)
        {
            showMessage("Error", "No records found");

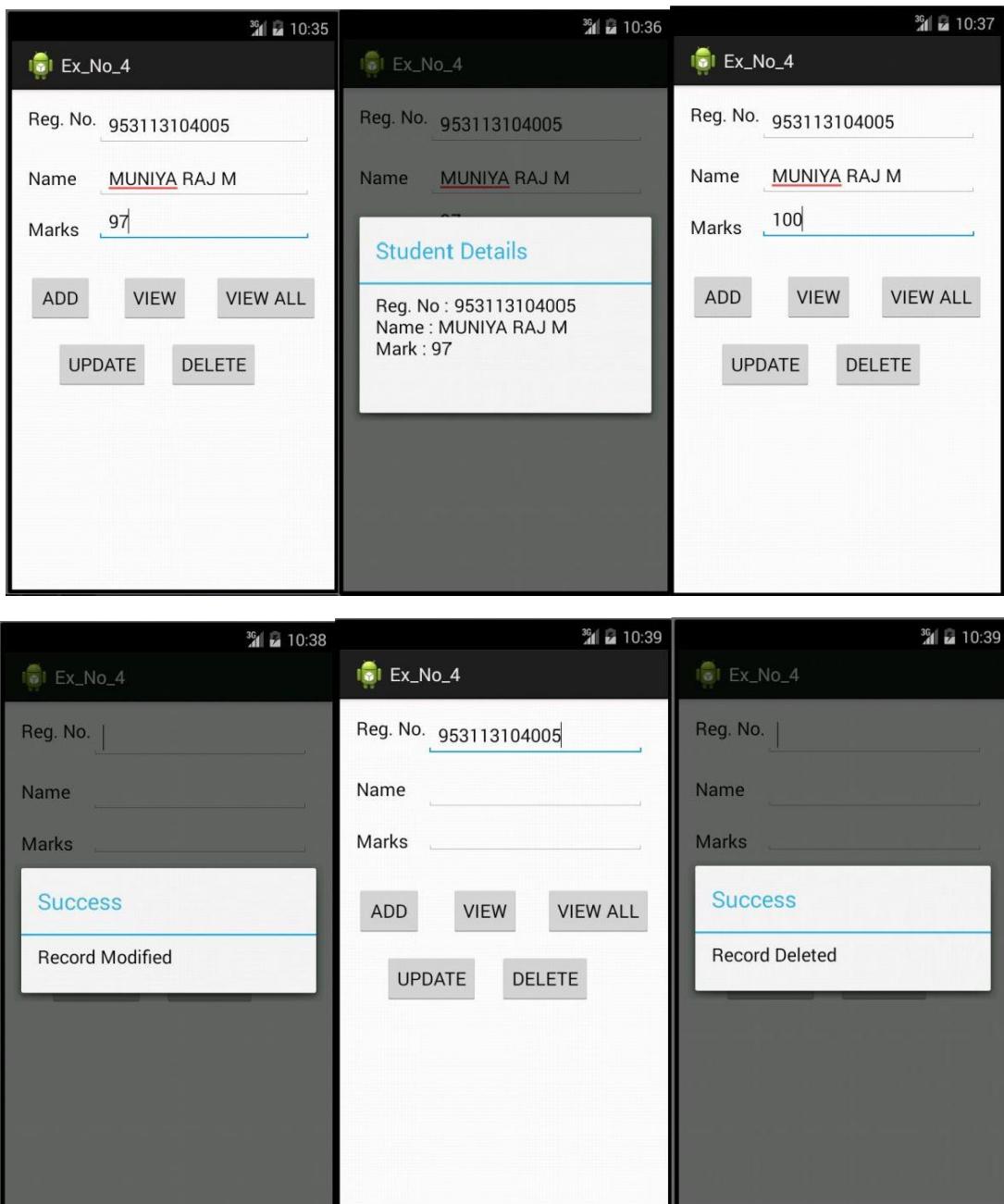
            return;
        }
        StringBuffer buffer=new StringBuffer();
        while(c.moveToNext())

        {
            buffer.append("Reg. No :
"+c.getString(0)+"\n");
            buffer.append("Name :
"+c.getString(1)+"\n");
            buffer.append("Mark :
"+c.getString(2)+"\n\n");
        }
        showMessage("Student Details",
                buffer.toString());
    }
});
}
public void showMessage(String title,String message)
{
    Builder builder=new
    Builder(this);
    builder.setCancelable
    (true);
    builder.setTitle(titl
e);
    builder.setMessage(me
ssage);
    builder.show();
}
public void clearText()

{
    regno.setText("");
    name.setText("");
    mark.setText("");
    regno.requestFocus();
}
}

```

OUTPUT:



RESULT:

Thus the application that makes use of database has been developed and the output was verified.

Ex. No: 6 Date:	Develop an application that makes use of RSS Feed
--------------------	--

AIM:

To develop an application that makes use of RSS Feed.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_8.
3. Go to package explorer in the left hand side. Select the project Ex_No_8.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Create the FrameLayout.
7. Create a new layout named as fragment_layout.xml which has following components:
 - a. ListView
 - b. ProgressBar
8. Create another one layout named as rss_item.xml which has only one TextView.
9. Again go to package explorer in the left hand side. Select the project Ex_No_7.
10. Go to src folder. Double click the MainActivity.java file.
11. In java file write the activities done by the application.
12. Create the following additional classes for this application:
 - a. Constants.java
 - b. PcWorldRssParser.java
 - c. RssAdapter.java
 - d. RssFragement.java
 - e. RssItem.java
 - f. RssService.java
13. Write appropriate actions for the created additional classes.
14. Get the following permission in AndroidManifest.xml file:
`<uses-permission android:name="android.permission.INTERNET" />`
15. Finally run the android application.

PROGRAMS:***activity_main.xml:***

```
<?xml version="1.0" encoding="utf-8"?>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:id="@+id/fragment_container"
    android:layout_height="fill_parent" />
```

fragement_layout.xml:

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <ListView android:id="@+id/listView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >
    </ListView>
    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true" />
</RelativeLayout>
```

rss_item.xml:

```
<?xml version="1.0" encoding="utf-8"?>

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/itemTitle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18dp"
    tools:ignore="SpUsage" />
```

MainActivity.java:

```
package com.example.ex_no_8;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
public class MainActivity extends FragmentActivity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
            addRssFragment();
        }
    }
}
```

```

    }
    private void addRssFragment() {

        FragmentManager manager = getSupportFragmentManager();
        FragmentTransaction transaction = manager.beginTransaction();
        RssFragment fragment = new RssFragment();
        transaction.add(R.id.fragment_container, fragment);
        transaction.commit();
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean("fragment_added", true);
    }
}

```

Constants.java

```

package com.example.ex_no_8;

public class Constants {
    public static final String TAG = "RssApp";
}

```

PcWorldRssParser.java

```

package com.example.ex_no_8;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import android.util.Xml;
public class PcWorldRssParser {
    // We don't use namespaces
    private final String ns = null;

    public List<RssItem> parse(InputStream inputStream) throws
        XmlPullParserException, IOException {
        try {
            XmlPullParser parser = Xml.newPullParser();
            parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
            parser.setInput(inputStream, null);
            parser.nextTag();
            return readFeed(parser);
        } finally {

```

```

        inputStream.close();
    }
}
private List<RssItem> readFeed(XmlPullParser parser) throws
    XmlPullParserException, IOException {
    parser.require(XmlPullParser.START_TAG, null, "rss");
    String title = null;
    String link = null;
    List<RssItem> items = new ArrayList<RssItem>();
    while (parser.next() != XmlPullParser.END_DOCUMENT) {

        if (parser.getEventType() != XmlPullParser.START_TAG) {

            continue;
        }
        String name = parser.getName();
        if (name.equals("title")) {
            title = readTitle(parser);
        } else if (name.equals("link")) {
            link = readLink(parser);
        }
        if (title != null && link != null) {

            RssItem item = new RssItem(title, link);
            items.add(item);
            title = null; link = null;
        }
    }
    return items;
}
private String readLink(XmlPullParser parser) throws XmlPullParserException,
    IOException
{
    parser.require(XmlPullParser.START_TAG, ns, "link");
    String link = readText(parser);
    parser.require(XmlPullParser.END_TAG, ns, "link");
    return link;
}
private String readTitle(XmlPullParser parser) throws
    XmlPullParserException, IOException {
    parser.require(XmlPullParser.START_TAG, ns, "title");
    String title = readText(parser);
    parser.require(XmlPullParser.END_TAG, ns, "title");
    return title;
}
// For the tags title and link, extract their text values.
private String readText(XmlPullParser parser) throws IOException,
    XmlPullParserException
{
    String result = "";
    if (parser.next() == XmlPullParser.TEXT) {
        result = parser.getText(); parser.nextTag();
    }
}

```

```

    return result;
}

}

```

RssAdapter.java

```

package com.example.ex_no_8;
import java.util.List;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
public class RssAdapter extends BaseAdapter {
    private final List<RssItem> items;
    private final Context context;
    public RssAdapter(Context context, List<RssItem> items) {
        this.items = items;
        this.context = context;
    }
    @Override
    public int getCount() {
        return items.size();
    }
    @Override
    public Object getItem(int position) {
        return items.get(position);
    }
    @Override
    public long getItemId(int id) {
        return id;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ViewHolder holder;
        if (convertView == null) {
            convertView = View.inflate(context, R.layout.rss_item, null);
            holder = new ViewHolder();
            holder.itemTitle = (TextView)
                convertView.findViewById(R.id.itemTitle);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolder) convertView.getTag();
        }
        holder.itemTitle.setText(items.get(position).getTitle());
        return convertView;
    }
}

class ViewHolder {
    TextView itemTitle;
}

```

```

    }
    static class ViewHolder {

        TextView itemTitle;

    }
}

```

RssFragement.java

```

package com.example.ex_no_8;
import java.util.List;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.ResultReceiver;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.Toast;
public class RssFragment extends Fragment implements OnItemClickListener {
    private ProgressBar
    progressBar; private
    ListView listView; private
    View view;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
        savedInstanceState) {
        if (view == null) {
            view = inflater.inflate(R.layout.fragment_layout,
            container, false);
            progressBar = (ProgressBar) view.findViewById(R.id.progressBar);
            listView = (ListView) view.findViewById(R.id.listView);
            listView.setOnItemClickListener(this);
            startService();
        } else {
            ViewGroup parent = (ViewGroup) view.getParent();
            parent.removeView(view);
        }
    }
}

```

```

        return view;
    }

    private void startService() {
        Intent intent = new Intent(getActivity(),
                RssService.class);
        intent.putExtra(RssService.RECEIVER,
                resultReceiver);
        getActivity().startService(intent);
    }

    private final ResultReceiver resultReceiver = new ResultReceiver(new
            Handler()) { @SuppressWarnings("unchecked")
            @Override
            protected void onReceiveResult(int resultCode, Bundle resultData) {
                progressBar.setVisibility(View.GONE);

                List<RssItem> items = (List<RssItem>)
resultData.getSerializable(RssService.ITEMS);
                if (items != null) {
                    RssAdapter adapter = new
                    RssAdapter(getActivity(), items);
                    listView.setAdapter(adapter);
                } else {
                    Toast.makeText(getActivity(), "An error occurred while
downloading the rss feed.",
                    Toast.LENGTH_LONG).show();
                }
            };
        };
        @Override
    }

    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        RssAdapter adapter = (RssAdapter) parent.getAdapter();
        RssItem item = (RssItem) adapter.getItem(position);
        Uri uri = Uri.parse(item.getLink());
        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
        startActivity(intent);
    }
}

```

RssItem.java

```

package com.example.ex_no_8;
public class RssItem {
    private final String title;
    private final String link;

```

```

public RssItem(String title, String link) {
    this.title = title;
    this.link = link;
}
public String getTitle() {
    return title;
}
public String getLink() {
    return link;
}
}

```

RssService.java

```

package com.example.ex_no_8;
import java.io.IOException;
import java.io.InputStream;
import java.io.Serializable;
import java.net.URL;
import java.util.List;
import org.xmlpull.v1.XmlPullParserException;
import android.app.IntentService;
import android.content.Intent;
import android.os.Bundle;
import android.os.ResultReceiver;
import android.util.Log;
public class RssService extends IntentService {
    private static final String RSS_LINK = "http://www.pcworld.com/index.rss";
    public static final String ITEMS = "items";
    public static final String RECEIVER = "receiver";
    public RssService() {
        super("RssService");
    }
    @Override
    protected void onHandleIntent(Intent intent) {
        Log.d(Constants.TAG, "Service started");
        List<RssItem> rssItems = null;
        try {
            PcWorldRssParser parser = new
            PcWorldRssParser();
            rssItems = parser.parse(getInputStream(RSS_LINK));
        }
        catch (XmlPullParserException e) {
            Log.w(e.getMessage(), e);
        }
        catch (IOException e) {
            Log.w(e.getMessage(), e);
        }
        Bundle bundle = new Bundle(); bundle.putSerializable(ITEMS,
                (Serializable) rssItems);
        ResultReceiver receiver = intent.getParcelableExtra(RECEIVER);
    }
}

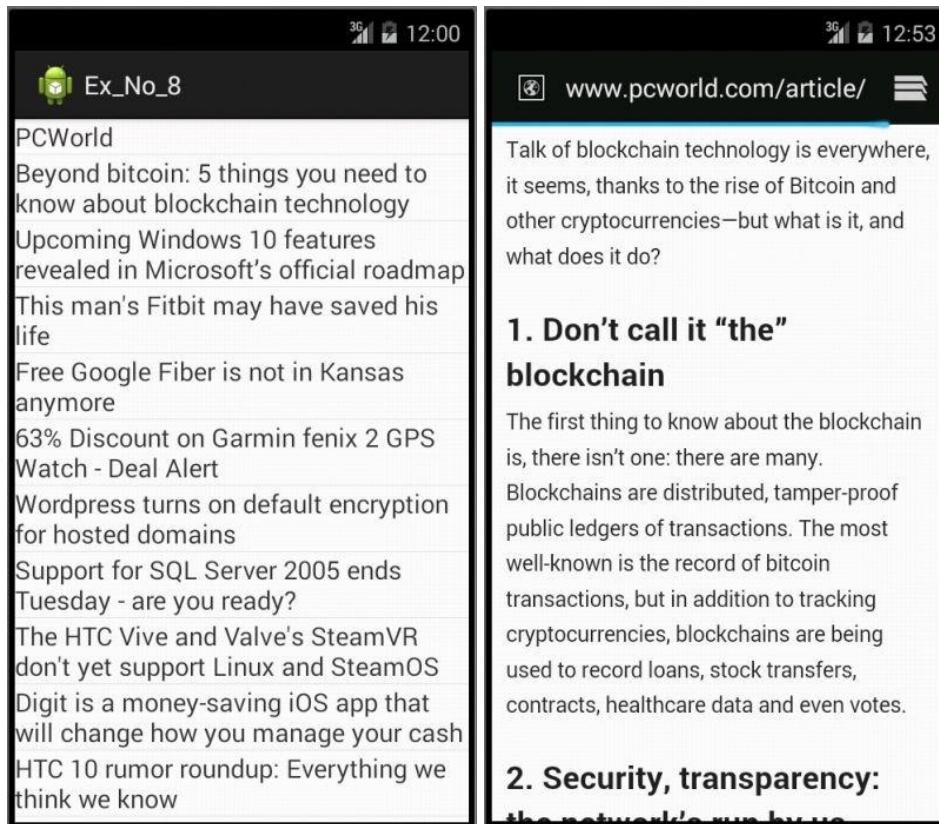
```

```

        receiver.send(0, bundle);
    }
    public InputStream getInputStream(String link) {
        try {
            URL url = new URL(link);
            return url.openConnection().getInputStream();
        }
        catch (IOException e) {
            Log.w(Constants.TAG, "Exception while retrieving the input
stream", e);
            return null;
        }
    }
}

```

OUTPUT:



RESULT:

Thus the application that makes use of RSS Feed has been developed and the output was verified.

Ex.No: 7	Implement an application that implements multi threading
Date :	

AIM:

To implement an application that implements multi threading.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_9.
3. Go to package explorer in the left hand side. Select the project Ex_No_9.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. One ProgressBar (Horizontal)
 - b. One Button with labeled as Start Progress
 - c. One TextView without any texts
7. Again go to package explorer in the left hand side. Select the project Ex_No_9.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as action of button.
10. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_9.MainActivity" >

    <ProgressBar
        android:id="@+id/progressBar1"
        ...>
```

```

        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/progressBar1"
    android:layout_centerHorizontal="true"
    android:text=" "
    android:textAppearance="?android:attr/textAppearanceLarge"
    tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:text="Start Progress"
    tools:ignore="HardcodedText"
    />

</RelativeLayout>

```

MainActivity.java:

```
package com.example.ex_no_9;
```

```

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar
; import android.widget.TextView;
public class MainActivity extends
    ActionBarActivity { @Override
    protected void onCreate(Bundle
        savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    final ProgressBar p=(ProgressBar)findViewById(R.id.progressBar1);

    final TextView
    t=(TextView)findViewById(R.id.textView1);
    Button b=(Button)findViewById(R.id.button1);
    b.setOnClickListener( new OnClickListener()
    {
        @Override
        public void onClick(View arg0) {
            // TODO Auto-generated method stub
        }
    });

    Runnable r=new Runnable(){ @Override
    public void run() {
        // TODO Auto-generated method stub
    }

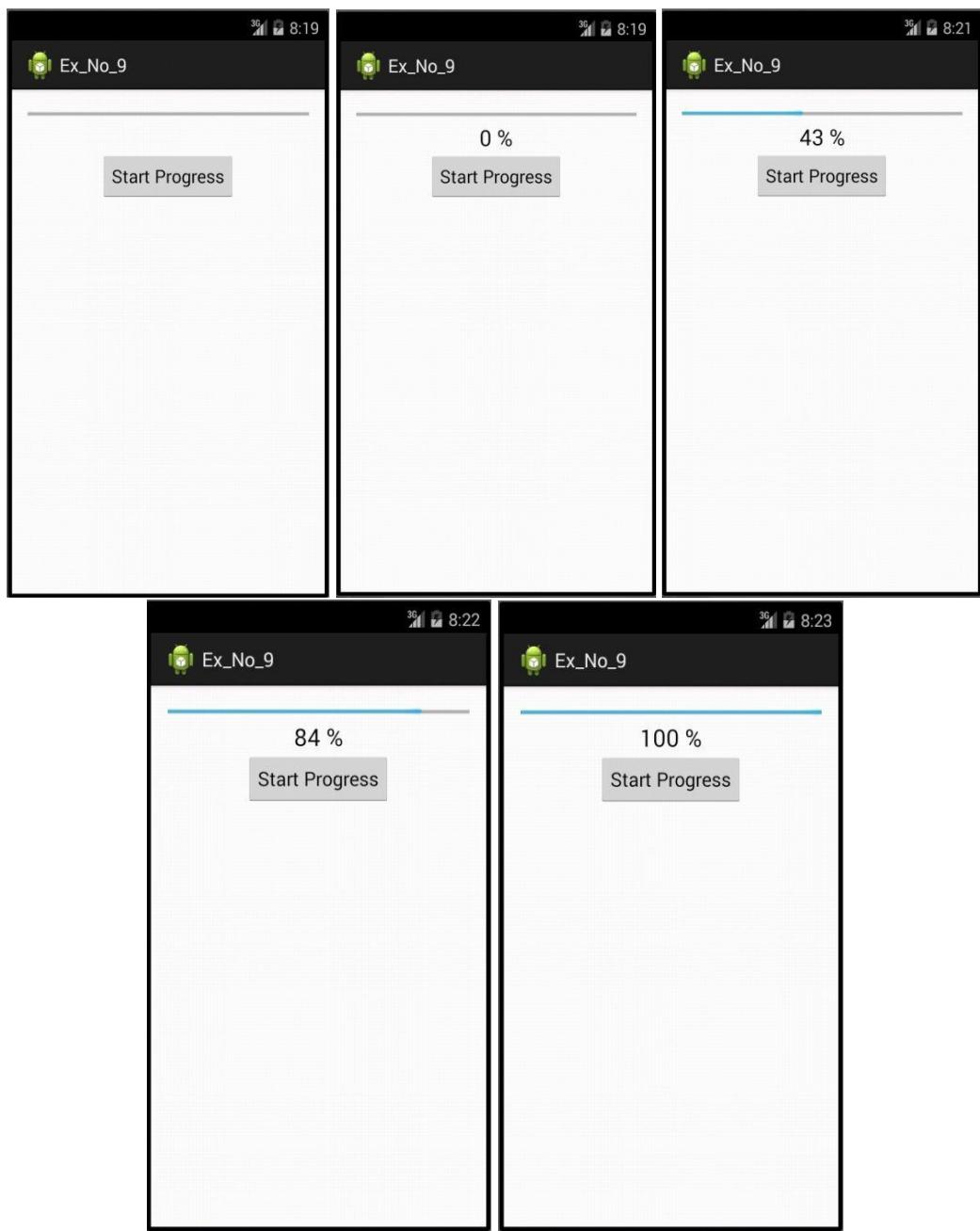
    for(int i=0;i<=100;i++)
    {
        final int temp=i;
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch
            e.printStackTrace();
        }
    }

    p.post(new Runnable()
    {
        @Override
    });
}
}

```

```
public void run() {  
    // TODO Auto-generated  
  
    p.setProgress(temp); t.setText(temp+" %");  
  
}  
  
}  
};  
new Thread(r).start();  
  
}  
});  
}  
}  
}
```

OUTPUT:



RESULT:

Thus the application that implements multi threading has been developed and the output was verified.

Ex.No: 8	Develop a native application that uses GPS location information
----------	--

AIM:

To develop a native application that uses GPS location information.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_5.
3. Go to package explorer in the left hand side. Select the project Ex_No_5.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. One TextView with text as Current Location
 - b. Two TextViews without any texts.
7. Again go to package explorer in the left hand side. Select the project Ex_No_5.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as finding current location and print them.
10. Get the following permission in AndroidManifest.xml file:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

11. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_5.MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```

        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="114dp"
        android:text=""
        android:textAppearance="?android:attr/textAppearanceMedium"
        tools:ignore="HardcodedText" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="51dp"
    android:text=""
    android:textAppearance="?android:attr/textAppearanceMedium"
    tools:ignore="HardcodedText" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="47dp"
    android:text="Current Location"
    android:textAppearance="?android:attr/textAppearanceLarge"
    tools:ignore="HardcodedText" />

</RelativeLayout>

```

MainActivity.java:

```

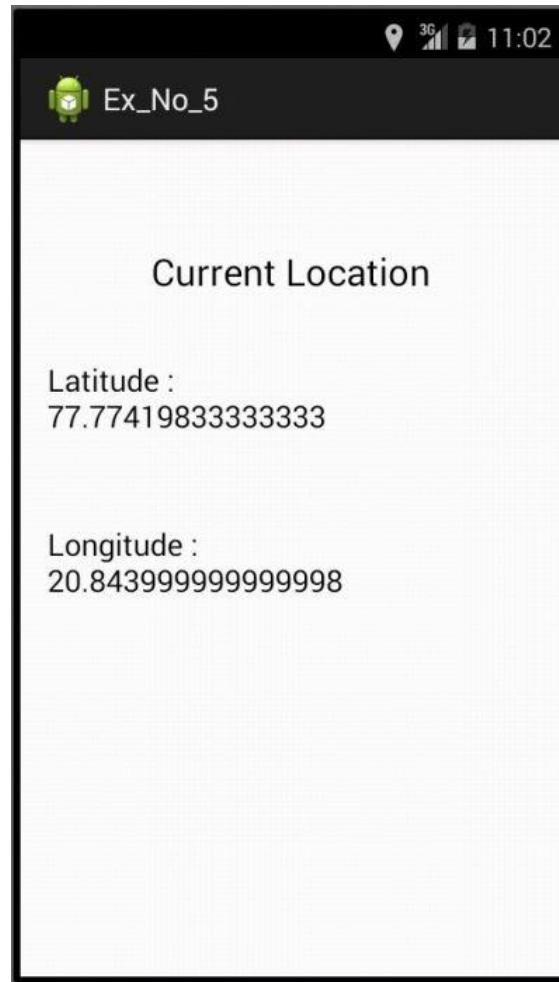
package com.example.ex_no_5;
import android.support.v7.app.ActionBarActivity;
import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;
public class MainActivity extends ActionBarActivity implements LocationListener{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        LocationManager
        lm=(LocationManager) getSystemService(Context.LOCATION_SERVICE);
    }
}

```

```

Criteria c=new Criteria();
String
s=lm.getBestProvider(c,
false); if(s!=null &&
!s.equals(""))
{
    Location
l=lm.getLastKnownLocation(s);
lm.requestLocationUpdates(s,
20000, 1, this); if(l!=null)
        onLocationChanged(l);
else
    Toast.makeText(getApplicationContext(), "Location can't be
retrieved !!!", Toast.LENGTH_LONG).show();
}
else
    Toast.makeText(getApplicationContext(), "Provider not found
!!!",
Toast.LENGTH_LONG).show();
}
@Override
public void onLocationChanged(Location arg0) {
    // TODO Auto-generated method stub
    TextView
t1=(TextView)findViewById(R.id.textVie
w1); t1.setText("Latitude :
\n"+arg0.getLatitude()); TextView
t2=(TextView)findViewById(R.id.textVie
w2); t2.setText("Longitude :
\n"+arg0.getLongitude());
}
@Override
public void onProviderDisabled(String arg0) {
    // TODO Auto-generated method stub
}
@Override
public void onProviderEnabled(String arg0) {
    // TODO Auto-generated method stub
}
@Override
public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
    // TODO Auto-generated method stub
}
}
}

```

OUTPUT:**RESULT:**

Thus the application that uses GPS location information has been developed and the output was verified.

Ex.No: 9	Implement an application that writes data to the SD card
Date : _____	

AIM:

To implement an application that writes data to the SD card.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_6.
3. Go to package explorer in the left hand side. Select the project Ex_No_6.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. Two EditTexts
 - b. Two Buttons with labeled as READ and SAVE
7. Again go to package explorer in the left hand side. Select the project Ex_No_6.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as actions of buttons.
10. Get the following permission in AndroidManifest.xml file:
`<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />`
11. Finally run the android application.

PROGRAMS:

activity_main.xml:

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_6.MainActivity" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"

```

```

        android:layout_alignParentTop="true"
        android:ems="10"
        android:hint="Path"
        tools:ignore="TextFields,HardcodedText" >

        <requestFocus />
</EditText>
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/editText1"
    android:layout_toRightOf="@+id/editText1"
    android:text="READ"
    tools:ignore="HardcodedText" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_centerVertical="true"
    android:ems="10"
    android:hint="Contents of File"
    android:inputType="textMultiLine"
    tools:ignore="HardcodedText" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_centerVertical="true"
    android:text="SAVE" tools:ignore="HardcodedText"
/>

</RelativeLayout>

```

MainActivity.java:

```

package
com.example.ex_no_6;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import android.support.v7.app.ActionBarActivity;
import android.annotation.SuppressLint;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

```

```

import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends ActionBarActivity {
    @SuppressLint("SdCardPath")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText e1=(EditText)findViewById(R.id.editText1);
        final EditText e2=(EditText)findViewById(R.id.editText2);
        Button
        b1=(Button)findViewById(R.id.button1); Button
        b2=(Button)findViewById(R.id.button2);
        String path=getPreferences(MODE_PRIVATE).getString("fpath",
        "/sdcard/file1"); e1.setText(path);
        b1.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0) {
                // TODO Auto-generated method stub

                File f=new
                File(e1.getText().toString()
                ); String s="";
                StringBuilder sb=new
                StringBuilder();
                FileReader fr = null;
                try {
                    fr = new FileReader(f);
                } catch (FileNotFoundException e) {
                    // TODO Auto-
                    generated catch block
                    e.printStackTrace();
                }
                BufferedReader br=new BufferedReader(fr);
                try {
                    while((s=br.readLine())!=null)
                    {
                        sb.append(
                        s+"\n");
                    }
                } catch (IOException e) {
                    // TODO Auto-
                    generated catch block
                    e.printStackTrace();
                }
                Toast.makeText(getApplicationContext(), "File Read Successfully !!!",
                Toast.LENGTH_LONG).show();
                e2.setText(sb);
            }
        });
    }
}

```

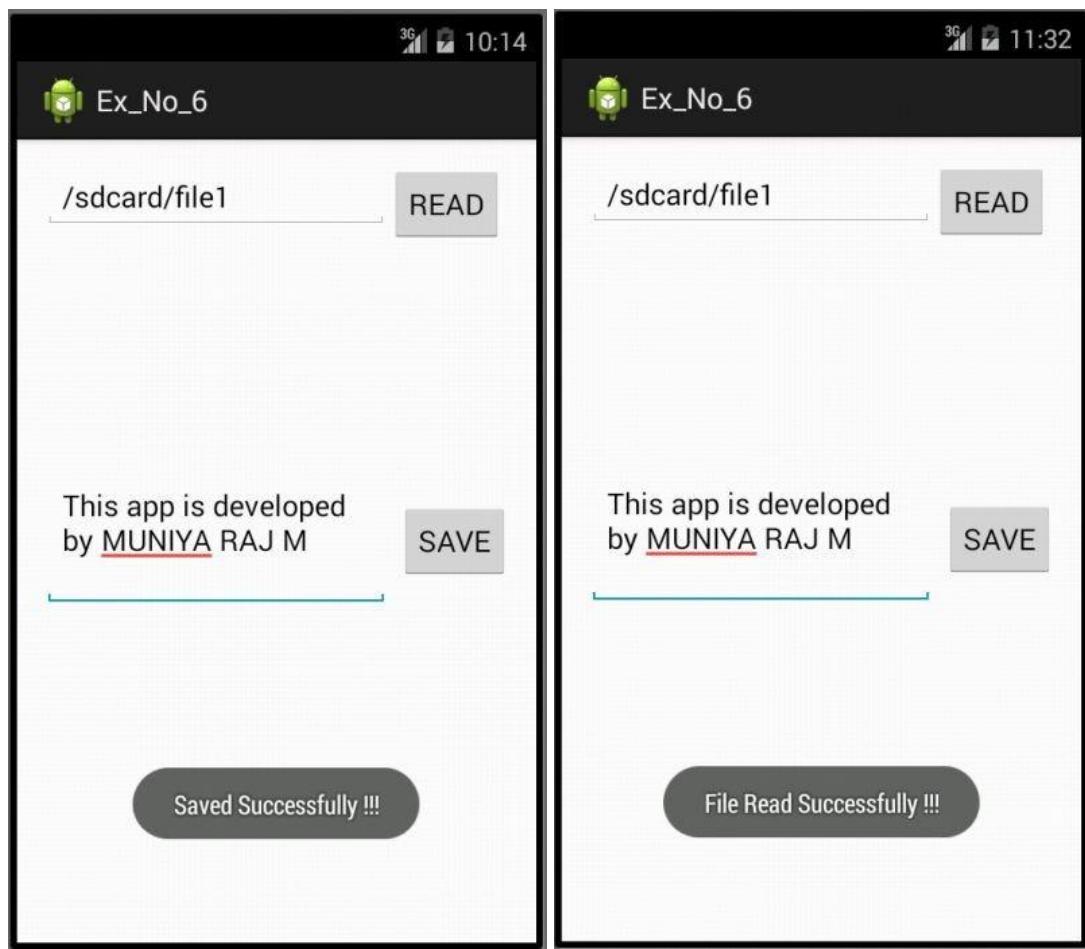
```

b2.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub

        File f=new File(e1.getText().toString());
        FileWriter fw = null;
        try {
            fw = new FileWriter(f);
        } catch (IOException e3){
            // TODO Auto-
            generated catch block
            e3.printStackTrace();
        }
        try {
            fw.write(e2.getText().toString());
        } catch (IOException e2){
            // TODO Auto-
            generated catch block
            e2.printStackTrace();
        }
        try {
            fw.close();
        } catch (IOException e2){
            // TODO Auto-generated catch block
            e2.printStackTrace();
        }
    }
    e=getPreferences(MODE_PRIVATE).edit();
    SharedPreferences.Editor.putString("fpath", f.getPath());
    e.commit();
    Toast.makeText(getApplicationContext(), "Saved Successfully !!!",
    Toast.LENGTH_LONG).show();
}
});
}
}

```

OUTPUT:



RESULT:

Thus the application that writes data to the SD card has been implemented and the output was verified.

Ex.No: 10	Implement an application that creates an alert upon receiving a message
-----------	--

AIM:

To implement an application that creates an alert upon receiving a message.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_10.
3. Go to package explorer in the left hand side. Select the project Ex_No_10.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. This application has no components, because this just generates a notification alone.
7. Again go to package explorer in the left hand side. Select the project Ex_No_10.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as receiving a message and notify it.
10. Get the following permissions in AndroidManifest.xml file:


```
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
```
11. Add Receiver class as receiver in AndroidManifest.xml file.
12. Finally run the android application.

PROGRAMS:

activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_10.MainActivity" >
</RelativeLayout>
```

MainActivity.java:

```
package com.example.ex_no_10;
import android.support.v7.app.ActionBarActivity;
import android.app.Notification;
import android.app.NotificationManager;
import android.content.Context;
```

```

import android.os.Bundle;
public class MainActivity extends ActionBarActivity {
    private static MainActivity inst;
    public static MainActivity instance() {
        // TODO Auto-generated method stub
        return inst;
    }
    public void onStart()
    {
        super.onStart();
        inst=this;
    }
    NotificationManager nm;
    Notification n;
    @SuppressWarnings("depr
ecation") @Override
    protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        nm=(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        n=new Notification(R.drawable.ic_launcher,"SMS Alert",System.currentTimeMillis());
    }
    @SuppressWarnings("deprecation")
    public void update_notification(String no, String msg) {
        // TODO Auto-generated method stub
        n.setLatestEventInfo(getApplicationContext(), no, msg, null);
        nm.notify(1337, n);
    }
}

```

Receiver.java:

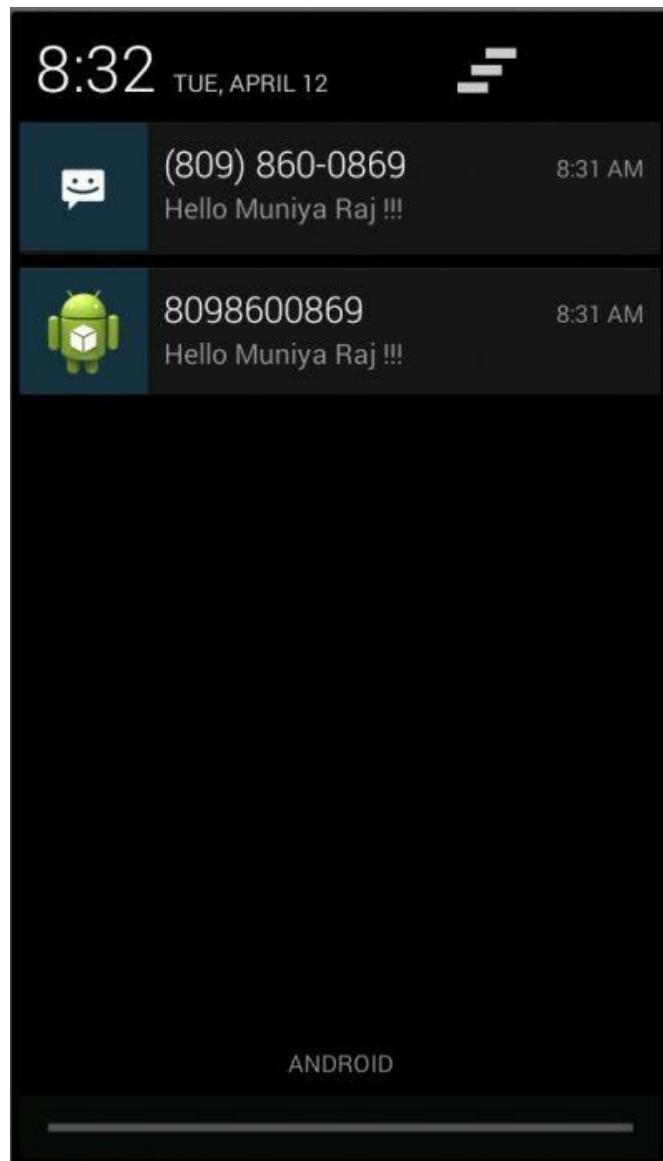
```

package com.example.ex_no_10;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
public class Receiver extends BroadcastReceiver { public static final
    String SMS_BUNDLE="pdus";
    @Override
    public void onReceive(Context arg0, Intent arg1) {
        // TODO Auto-generated
        method stub String no =
        null,msg = null; Bundle
        b=arg1.getExtras();
        if(b!=null)
        {
            Object[] sms=(Object[])b.get(SMS_BUNDLE);
            for(int i=0;i<sms.length;++i)
            {
                SmsMessage
                sm=SmsMessage.createFromPdu((byte[])sms[i]);
                no=sm.getOriginatingAddress();
            }
        }
    }
}

```

```
        msg=sm.getMessageBody().toString();
    }
MainActivity inst=MainActivity.instance();
inst.update_notification(no,msg);
}
}
```

OUTPUT:



RESULT:

Thus the application that creates an alert upon receiving a message has been developed and the output was verified.

Ex.No: 11	Write a mobile application that creates alarm clock
Date :	

AIM:

To implement an application that creates alarm clock.

PROCEDURE:

1. Open Eclipse IDE.
2. Create the project Ex_No_11.
3. Go to package explorer in the left hand side. Select the project Ex_No_11.
4. Go to res folder and select layout. Double click the activity_main.xml file.
5. Now you can see the Graphical layout window.
6. Drag and drop the following components:
 - a. DatePicker
 - b. TimePicker
 - c. Button with labeled as SET ALARM
7. Again go to package explorer in the left hand side. Select the project Ex_No_11.
8. Go to src folder. Double click the MainActivity.java file.
9. In java file write the activities done by the application such as notify the alarm.
10. Get the following permission in AndroidManifest.xml file:
`<uses-permission android:name="android.permission.WAKE_LOCK"/>`
11. Add Alarm class as a receiver in AndroidManifest.xml file.
12. Finally run the android application.

PROGRAMS:

activity_main.xml:

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ex_no_11.MainActivity" >
    <DatePicker
        android:id="@+id/datePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true" />
    <TimePicker
        android:id="@+id/timePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/datePicker1"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_marginBottom="71dp" />

```

```

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/timePicker1"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_marginBottom="14dp"
    android:text="SET ALARM"
    tools:ignore="HardcodedText" />
</RelativeLayout>

```

MainActivity.java:

```

package com.example.ex_no_11;
import java.util.Calendar;
import android.support.v7.app.ActionBarActivity;
import android.app.AlarmManager;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TimePicker;
import android.widget.Toast;
public class MainActivity extends ActionBarActivity {
    private static MainActivity inst;
    public static MainActivity instance() {
        // TODO Auto-generated method stub
        return inst;
    }
    public void onStart()
    {
        super.onStart();
        inst=this;
    }
    NotificationManager nm;
    Notification n;
    @SuppressWarnings("deprecation")
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TimePicker tp=(TimePicker)findViewById(R.id.timePicker1);

        final DatePicker dp=(DatePicker)findViewById(R.id.datePicker1);
        Button b=(Button)findViewById(R.id.button1);
    }
}

```

```

nm=(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
n=new Notification(R.drawable.ic_launcher,"ALARM",System.currentTimeMillis());

tp.setIs24HourView(false);
Calendar now=Calendar.getInstance(); dp.init(now.get(Calendar.YEAR),
now.get(Calendar.MONTH),
now.get(Calendar.DAY_OF_MONTH),null);
tp.setCurrentHour(now.get(Calendar.HOUR_OF_DAY));
tp.setCurrentMinute(now.get(Calendar.MINUTE));
b.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub

        Calendar current=Calendar.getInstance();
        Calendar alarm=Calendar.getInstance();
        alarm.set(dp.getYear(), dp.getMonth(),
dp.getDayOfMonth(), tp.getCurrentHour(), tp.getCurrentMinute(), 00);
        if(alarm.compareTo(current)<=0)
            Toast.makeText(getApplicationContext(),
"Invalid Date and Time !!!",
Toast.LENGTH_LONG).show();
        else
        {
            Intent i=new
Intent(MainActivity.this,Alarm.class);
PendingIntent pi=PendingIntent.getBroadcast(MainActivity.this, 123, i, 0);

        AlarmManager am=(AlarmManager) getSystemService(ALARM_SERVICE);
        am.set(AlarmManager.RTC_WAKEUP,alarm.getTimeInMillis(), pi);

        Set ON !!!", Toast.LENGTH_LONG).show();
    }
}
});;
}
@SuppressWarnings("deprecation")
Toast.makeText(getApplicationContext(), "Alarm is
public void update_notification(String no, String msg){
    // TODO Auto-generated method stub
    n.setLatestEventInfo(getApplicationContext(),
no, msg, null); nm.notify(1337, n);
}
}


```

Alarm.java:

```

package com.example.ex_no_11;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class Alarm extends BroadcastReceiver{

```

```

@Override
public void onReceive(Context arg0, Intent arg1) {
    // TODO Auto-generated method
    stub MainActivity
    inst=MainActivity.instance();
    inst.update_notification("Alarm","Wake up ! Wake up !!");
}
}

```

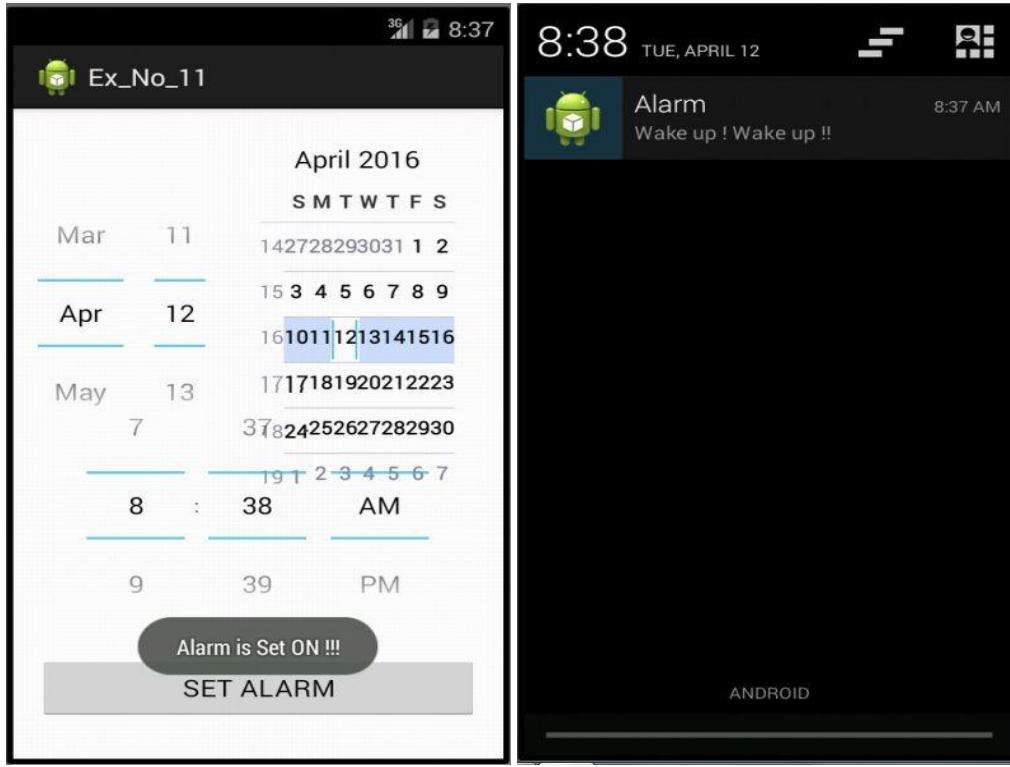
AndroidManifest.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ex_no_11"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".Alarm" />
    </application>
</manifest>

```

OUTPUT:



RESULT:

Thus the application that creates an alert upon receiving a message has been developed and the output was verified.

CHAPTER 9

IMPORTANT QUESTIONS

9.1 SHORT QUESTIONS AND ANSWERS

1) What is Android?

It is an open-sourced operating system that is used primarily on mobile devices, such as cell phones and tablets. It is a Linux kernel-based system that's been equipped with rich components that allows developers to create and run apps that can perform both basic and advanced functions.

2) What Is the Google Android SDK?

The Google Android SDK is a toolset that developers need in order to write apps on Android enabled devices. It contains a graphical interface that emulates an Android driven handheld environment, allowing them to test and debug their codes.

3) What is the Android Architecture?

Android Architecture is made up of 4 key components:

- Linux Kernel
- Libraries
- Android Framework
- Android Applications

4) Describe the Android Framework.

The Android Framework is an important aspect of the Android Architecture. Here you can find all the classes and methods that developers would need in order to write applications on the Android environment.

5) What is AAPT?

AAPT is short for Android Asset Packaging Tool. This tool provides developers with the ability to deal with zip-compatible archives, which includes creating, extracting as well as viewing its contents.

6) What is the importance of having an emulator within the Android environment?

The emulator lets developers "play" around an interface that acts as if it were an actual mobile device. They can write and test codes, and even debug. Emulators are a safe place for testing codes especially if it is in the early design phase.

7) What is the use of an activityCreator?

An activityCreator is the first step towards the creation of a new Android project. It is made up of a shell script that will be used to create new file system structure necessary for writing codes within the Android IDE.

8) Describe Activities.

Activities are what you refer to as the window to a user interface. Just as you create windows in order to display output or to ask for an input in the form of dialog

boxes, activities play the same role, though it may not always be in the form of a user interface.

9) What are Intents?

Intents displays notification messages to the user from within the Android enabled device. It can be used to alert the user of a particular state that occurred. Users can be made to respond to intents.

10) Differentiate Activities from Services.

Activities can be closed, or terminated anytime the user wishes. On the other hand, services are designed to run behind the scenes, and can act independently. Most services run continuously, regardless of whether there are certain or no activities being executed.

11) What items are important in every Android project?

These are the essential items that are present each time an Android project is created:

- AndroidManifest.xml
- build.xml
- bin/
- src/
- res/
- assets/

12) What is the importance of XML-based layouts?

The use of XML-based layouts provides a consistent and somewhat standard means of setting GUI definition format. In common practice, layout details are placed in XML files while other items are placed in source files.

13) What are containers?

Containers, as the name itself implies, holds objects and widgets together, depending on which specific items are needed and in what particular arrangement that is wanted. Containers may hold labels, fields, buttons, or even child containers, as examples.

14) What is Orientation?

Orientation, which can be set using setOrientation(), dictates if the LinearLayout is represented as a row or as a column. Values are set as either HORIZONTAL or VERTICAL.

15) What is the importance of Android in the mobile market?

Developers can write and register apps that will specifically run under the Android environment. This means that every mobile device that is Android enabled will be able to support and run these apps. With the growing popularity of Android mobile devices, developers can take advantage of this trend by creating and uploading their apps on the Android Market for distribution to anyone who wants to download it.

16) What do you think are some disadvantages of Android?

Given that Android is an open-source platform, and the fact that different Android operating systems have been released on different mobile devices, there's no clear cut policy to how applications can adapt with various OS versions and upgrades.

One app that runs on this particular version of Android OS may or may not run on another version. Another disadvantage is that since mobile devices such as phones and tabs come in different sizes and forms, it poses a challenge for developers to create apps that can adjust correctly to the right screen size and other varying features and specs.

17) What is adb?

Adb is short for Android Debug Bridge. It allows developers the power to execute remote shell commands. Its basic function is to allow and control communication towards and from the emulator port.

18) What are the four essential states of an activity?

- Active – if the activity is at the foreground
- Paused – if the activity is at the background and still visible
- Stopped – if the activity is not visible and therefore is hidden or obscured by another activity
- Destroyed – when the activity process is killed or completed terminated

19) What is ANR?

ANR is short for Application Not Responding. This is actually a dialog that appears to the user whenever an application have been unresponsive for a long period of time.

20) Which elements can occur only once and must be present?

Among the different elements, the and elements must be present and can occur only once. The rest are optional, and can occur as many times as needed.

21) How are escape characters used as attribute?

Escape characters are preceded by double backslashes. For example, a newline character is created using '\\n'

22) What is the importance of settings permissions in app development?

Permissions allow certain restrictions to be imposed primarily to protect data and code. Without these, codes could be compromised, resulting to defects in functionality.

23) What is the function of an intent filter?

Because every component needs to indicate which intents they can respond to, intent filters are used to filter out intents that these components are willing to receive. One or more intent filters are possible, depending on the services and activities that is going to make use of it.

24) Enumerate the three key loops when monitoring an activity

- Entire lifetime – activity happens between onCreate and onDestroy
- Visible lifetime – activity happens between onStart and onStop
- Foreground lifetime – activity happens between onResume and onPause

25) When is the onStop() method invoked?

A call to onStop method happens when an activity is no longer visible to the user, either because another activity has taken over or if in front of that activity.

26) Is there a case wherein other qualifiers in multiple resources take precedence over locale?

Yes, there are actually instances wherein some qualifiers can take precedence over locale. There are two known exceptions, which are the MCC (mobile country code) and MNC (mobile network code) qualifiers.

27) What are the different states wherein a process is based?

There are 4 possible states:

- foreground activity
- visible activity
- background activity
- empty process

28) How can the ANR be prevented?

One technique that prevents the Android system from concluding a code that has been responsive for a long period of time is to create a child thread. Within the child thread, most of the actual workings of the codes can be placed, so that the main thread runs with minimal periods of unresponsive times.

29) What role does Dalvik play in Android development?

Dalvik serves as a virtual machine, and it is where every Android application runs. Through Dalvik, a device is able to execute multiple virtual machines efficiently through better memory management.

30) What is the AndroidManifest.xml?

This file is essential in every application. It is declared in the root directory and contains information about the application that the Android system must know before the codes can be executed.

31) What is the proper way of setting up an Android-powered device for app development?

The following are steps to be followed prior to actual application development in an Android-powered device:

- Declare your application as "debuggable" in your Android Manifest.
- Turn on "USB Debugging" on your device.
- Set up your system to detect your device.

32) Enumerate the steps in creating a bounded service through AIDL.

1. create the .aidl file, which defines the programming interface
2. implement the interface, which involves extending the inner abstract Stub class as well as implanting its methods.
3. expose the interface, which involves implementing the service to the clients.

33) What is the importance of Default Resources?

When default resources, which contain default strings and files, are not present, an error will occur and the app will not run. Resources are placed in specially named subdirectories under the project res/ directory.

34) When dealing with multiple resources, which one takes precedence?

Assuming that all of these multiple resources are able to match the configuration of a device, the 'locale' qualifier almost always takes the highest precedence over the others.

35) When does ANR occur?

The ANR dialog is displayed to the user based on two possible conditions. One is when there is no response to an input event within 5 seconds, and the other is when a broadcast receiver is not done executing within 10 seconds.

36) What is AIDL?

AIDL, or Android Interface Definition Language, handles the interface requirements between a client and a service so both can communicate at the same level through interprocess communication or IPC. This process involves breaking down objects into primitives that Android can understand. This part is required simply because a process cannot access the memory of the other process.

37) What data types are supported by AIDL?

AIDL has support for the following data types:

- string
- charSequence
- List
- Map
- all native Java data types like int,long, char and Boolean

38) What is a Fragment?

A fragment is a part or portion of an activity. It is modular in a sense that you can move around or combine with other fragments in a single activity. Fragments are also reusable.

39) What is a visible activity?

A visible activity is one that sits behind a foreground dialog. It is actually visible to the user, but not necessarily being in the foreground itself.

40) When is the best time to kill a foreground activity?

The foreground activity, being the most important among the other states, is only killed or terminated as a last resort, especially if it is already consuming too much memory. When a memory paging state has been reached by a foreground activity, then it is killed so that the user interface can retain its responsiveness to the user.

41) Is it possible to use or add a fragment without using a user interface?

Yes, it is possible to do that, such as when you want to create a background behavior for a particular activity. You can do this by using add(Fragment,string) method to add a fragment from the activity.

42) How do you remove icons and widgets from the main screen of the Android device?

To remove an icon or shortcut, press and hold that icon. You then drag it downwards to the lower part of the screen where a remove button appears.

43) What are the core components under the Android application architecture?

There are 5 key components under the Android application architecture:

- services
- intent
- resource externalization

- notifications
- content providers

44) What composes a typical Android application project?

A project under Android development, upon compilation, becomes an .apk file. This apk file format is actually made up of the AndroidManifest.xml file, application code, resource files, and other related files.

45) What is a Sticky Intent?

A Sticky Intent is a broadcast from sendStickyBroadcast() method such that the intent floats around even after the broadcast, allowing others to collect data from it.

46) Do all mobile phones support the latest Android operating system?

Some Android-powered phone allows you to upgrade to the higher Android operating system version. However, not all upgrades would allow you to get the latest version. It depends largely on the capability and specs of the phone, whether it can support the newer features available under the latest Android version.

47) What is portable wi-fi hotspot?

Portable Wi-Fi Hotspot allows you to share your mobile internet connection to other wireless device. For example, using your Android-powered phone as a Wi-Fi Hotspot, you can use your laptop to connect to the Internet using that access point.

48) What is an action?

In Android development, an action is what the intent sender wants to do or expected to get as a response. Most application functionality is based on the intended action.

49) What is the difference between a regular bitmap and a nine-patch image?

In general, a Nine-patch image allows resizing that can be used as background or other image size requirements for the target device. The Nine-patch refers to the way you can resize the image: 4 corners that are unscaled, 4 edges that are scaled in 1 axis, and the middle one that can be scaled into both axes.

50) What language is supported by Android for application development?

The main language supported is Java programming language. Java is the most popular language for app development, which makes it ideal even for new Android developers to quickly learn to create and deploy applications in the Android environment.

ABOUT AUTHORS

John T Mesia Dhas received his Ph.D. in Computer Science and Engineering from Vel Tech University, Chennai, India. He has 16 years of Experience in the field of Education and Industry, currently he is working as an Associate Professor with Computer Science and Engineering Department of Audisankara College of Engineering and Technology, Gudur, Andhra Pradesh, India under Jawaharlal Nehru Technological University Anantapuramu.

He is also doing researches in Software Engineering and Data Analytics fields. He has published more than 25 research articles in conferences and Journals.

S. Naveen Kumar received his Ph.D. in Computer Science and Engineering from Annamalai University, Chidamparam, India. He has 4 years of Experience in the field of Education and Industry, currently he is working as an Associate Professor with Computer Science and Engineering Department of Audisankara College of Engineering and Technology, Gudur, Andhra Pradesh, India under Jawaharlal Nehru Technological University Anantapuramu.

He is also doing researches in Computer Networks, Mobile Securities and Artificial Intelligence fields. He has published more than 15 research articles in conferences and Journals.

D. Surendra received his M. Tech. in Computer Science and Engineering from Jawaharlal Nehru Technological University Anantapuramu, India. He has 12 years of Experience in the field of Education and Industry, currently he is working as an Assistant Professor with Computer Science and Engineering Department of Audisankara College of Engineering and Technology, Gudur, Andhra Pradesh, India under Jawaharlal Nehru Technological University Anantapuramu.

He is also doing researches in Network Securities, Mobile Application Development and Machine Learning fields. He has published more than 10 research articles in conferences and Journals.

ISBN: 978-93-5445-403-5

Price:

OTHER BOOKS

S. No	Title	ISBN
1	C LOGIC PROGRAMMING	978-93-5416-366-1
2	MODERN METRICS (MM): THE FUNCTIONAL SIZE ESTIMATOR FOR MODERN SOFTWARE	978-93-5408-510-9
3	PYTHON 3.7.1 Vol - I	978-93-5416-045-5
4	SOFTWARE SIZING APPROACHES	978-93-5437-820-1
5	DBMS PRACTICAL PROGRAMS	978-93-5437-572-9
6	SERVICE ORIENTED ARCHITECTURE	978-93-5416-496-5
7	ANDROID APPLICATIONS DEVELOPMENT PRACTICAL APPROACH	978-93-5445-403-5

For free E-Books: jtdhasres@gmail.com

ISBN: 978-93-5445-403-5

Price: